División de Ciencias Básicas e Ingeniería

Anticoloraciones en gráficas

Tesis en requerimiento parcial para obtener el grado de

Doctor en Optimización
por
M. en O. Luis Eduardo Urbán Rivero

Asesores:
Dr. Javier Ramírez Rodríguez
Departamento de Sistemas, UAM Azcapotzalco

Dr. Rafael López Bracho
Departamento de Sistemas, UAM Azcapotzalco

14 de junio de 2018

Dedicatoria

Para ti mamá, no se me ocurre mejor manera de decirte que te quiero y nunca lo he dejado de hacer.

A la memoria de ese hijo que no pudo ser, en parte logré esto por que tu no estás, espero algún día me perdones.

A la memoria de mi hermana, donde quiera que estés espero algún día reunirme contigo.

Agradecimientos

Los agradecimientos los quiero separar en tres partes, primero los personales, luego los técnicos y al final los especiales por su relevancia en mi crecimiento como investigador.

Personales

A mi madre y hermana por estar siempre a mi lado, aunque lejos pero se que ahí siguen.

A Wendy por estar a mi lado durante esta etapa a pesar de todo.

A Franco Zamora Aguilar por su apoyo.

A mis abuelos porque son el otro bastión de paternidad en mi vida.

A mi tío Omar y mi tía Angeles por ser las únicas personas que nunca me han abandonado.

A todos mis compañeros y compañeras del posgrado por apoyarme cuando lo necesité en especial a Gualberto, a Alberto (Betito) y a Sergio.

Al ahora Dr. Rodrigo Alexander Castro Campos por su ayuda y palabras sinceras.

Al Dr. Risto Rangel Kuoppa por su apoyo incondicional.

A Magnolia porque simplemente le debo una.

A todos los profesores del posgrado que siempre estuvieron al pendiente de mi trabajo y de mi persona.

Técnicos

A mis asesores Dr. Javier Ramírez Rodríguez y Dr. Rafael López Bracho por hacer esto posible y por su paciencia.

A los sinodales por su tiempo y consejos sobre la tesis.

A CONACYT y al pueblo Mexicano ya que con su trabajo permitieron que yo alcanzara esta meta.

Especiales

Al Dr. Francisco Javier Zaragoza Martínez por todo.

En general gracias a todas las personas que ademas de lograr que esto sea posible han formado parte de mi vida de una u otra forma. Gracias por hacer de mi mundo un lugar mejor a pesar de todas las contrariedades.

Resumen

Uno de los problemas más conocidos y estudiados de la teoría de gráficas es el problema de coloración. Un caso especial del problema de coloración supuso una de las preguntas matemáticas más controvertidas de la humanidad, conocido ahora como el teorema de los cuatro colores. Controvertida por dos razones, la dificultad para encontrar su respuesta y porque en su demostración se emplearon por primera vez computadoras.

En el mismo sentido de los problemas de coloración, Berge propuso en los 70's un problema que 30 años después provocaría la creación de una teoría bajo el nombre de el problema de anticoloración. El problema de Berge versa sobre lo siguiente. Dado un tablero de ajedrez de $n \times n$, b reinas negras y w reinas blancas. ¿Es posible colocar las reinas negras y blancas en el tablero sin que se ataquen mutuamente? En las anticoloraciones se le pide a la coloración la condición opuesta a la coloración clásica, es decir si dos vértices son adyacentes estos deben tener el mismo color. Para que esto tenga sentido se deben agregar algunas condiciones al problema, como la existencia de vértices incoloros y que las cardinalidades de las clases de color sean fijas.

En esta tesis se presenta la demostración de una conjetura que deriva del problema de Berge con caballos en lugar de reinas. Además de que se proponen modelos de programación lineal entera, no lineal entera y multiobjetivo para poder resolver problemas de anticoloración con dos colores por medio de algoritmos exactos. Además de lo anterior se presenta una mejora de las heurísticas parados colores y se propone una nueva heurística basada en algoritmos genéticos. Por último se dan las extensiones necesarias para trabajar el problema de anticoloración con más de dos colores, tanto con algoritmos exactos como con métodos heurísticos.

Introducción

El concepto de anticoloración surge como una generalización del concepto de separador de vértices (VSP por sus siglas en inglés). Formalmente una anticoloración se define como sigue.

Definición (Anticoloración) *Una anticoloración de los vértices de una gráfica $G = (V, E)$ es una coloración parcial (no es necesario que todos tengan color) $a : V \to S \cup \{incoloro\}$, con S el conjunto de colores. Tal que para cada $\{vw\} \in E$, $a(v) = a(w)$ o al menos uno de los dos es incoloro.*

La principal característica de la anticoloración es que se pide que vértices adyacentes tengan asignados colores iguales. Al ser una coloración parcial, implica que no es necesario colorear todos los vértices. Con tan sólo la definición anterior no hay un problema que atacar, puesto que bastaría con asignar un único color a los vértices de la gráfica para tener una anticoloración valida. El problema de anticoloración (PAC) pide como parámetro de entrada las cardinalidades de las clases de color, es decir se pide a priori el número de vértices que deben ser coloreados con cada color. Formalmente el PAC se define como sigue.

Problema (PAC)
Entrada: *Una gráfica $G = (V, E)$ y $c_1, c_2, \ldots, c_k$ enteros positivos.*
Salida: *1 si se pueden colorear c_1 vértices del color 1, c_2 vértices del color 2 y así sucesivamente en G bajo la condición de anticoloración, 0 en otro caso.*

Se sabe que el PAC es NP-Completo aun para dos colores. En la literatura se ha abordado el caso de dos colores conocido como el Problema de coloración blanco y negro (BWC por sus siglas en inglés). El capítulo 1 contiene la notación, los conceptos y el marco teórico para entender esta tesis.

El capítulo 2 contiene el estado actual del conocimiento del tema de anticoloración de gráficas. La investigación actual en anticoloraciones se puede resumir en dos vertientes.

1. Algoritmos exactos.

2. Métodos heurísticos.

En el caso de los algoritmos exactos se han encontrado algunas familias de gráficas donde decidir si se puede o no anticolorear dicha familia se puede hacer en tiempo polinomial así como los algoritmos para lograr este objetivo. Las familias de gráficas que han sido exploradas son árboles, gráficas cordales y las relacionadas con tableros de ajedrez.

En el caso de los métodos heurísticos, el BWC ha sido abordado con búsqueda tabú y GRASP, siendo estos los únicos puntos de comparación.

Las aportaciones de esta tesis están divididas de la misma manera en la que han sido presentadas las que se encuentran en el estado actual del conocimiento.

En el capítulo 3 se presentan las aportaciones de esta tesis en cuanto a algoritmos exactos. Se presenta un programa entero que permite tener una restricción de anticoloración por vértice, en contraste con los modelos actuales que tienen una restricción de anticoloración por arista lo cual reduce el tamaño del modelo. También se presenta la generalización del modelo para más de dos colores. Adicional al programa entero se presenta un programa cuadrático de factibilidad tanto para dos colores como para más de dos colores.

En relación a las familias de gráficas cuyas anticoloraciones se pueden obtener en tiempo polinomial, se resolvió el siguiente problema planteado por Berend *et al.* en [5]:

Problema (BWC-Balanceado en tableros de ajedrez)
Entrada: *Un tablero de ajedrez de $m \times n$, la pieza del tipo $t \in \{rey, reina, caballo, torre, alfil\}$.*
Salida: $\phi_t(m,n) = \max\{\min\{b,w\}\}$*, bajo la condición de anticoloración.*

Donde b es el número de piezas negras y w es el número de piezas blancas. Se presentan dos casos. El primero es la generalización rectangular de tableros de reyes junto con las fórmulas explicitas el valor de $\phi_{rey}(m,n)$, además de la demostración de una conjetura propuesta por Berend *et al.*, para el valor de $\phi_{caballo}(m,n)$. Por último se extiende la manera de resolver el PAC para más de dos colores a partir de subproblemas de sólo dos colores, se muestra que esto no implica optimalidad en todos los casos, pero de momento es la única forma de abordar el problema adicional a la del modelo de más de dos colores.

El capítulo 4 presenta las aportaciones en relación a los métodos heurísticos. Se realizó una calibración automática de un método conocido como GRASP con el que se trató el BWC en [6] y adicional a esto, el método se hizo multiarranque para observar si experimentalmente tenía alguna diferencia en cuanto a las soluciones encontradas por el GRASP original y el calibrado automáticamente. Se desarrolló un algoritmo genético y se compara su desempeño con respecto a la búsqueda tabú que es hasta este trabajo, el punto de referencia. Para el algoritmo genético se tomaron en cuenta a los principales métodos de selección, cruza, mutación y las combinaciones de ellos. Así como el GRASP se calibró de manera automática, también se realizó el mismo procedimiento con el algoritmo genético pero agregando la calibración de los operadores del algoritmo. Se presentan dos variantes del algoritmo genético. La primera con tasa de mutación constante y la segunda con tasa de mutación variable, siendo esta última la que mejor desempeño mostró.

Por último se presenta una nueva forma de abordar el PAC como un problema multiobjetivo en donde una solución a los modelos exactos ya antes mencionados puede ser vista como

una solución no dominada. En este caso se utilizó el programa cuadrático y una escalarización de Chebyshev para encontrar el frente aproximado de Pareto. Lo anterior no implica la disminución en la complejidad del problema pero sí una nueva manera de abordarlo que no había sido estudiada hasta el momento y que también resulta útil cuando se trata del PAC con más de dos colores.

Las contribuciones de esta tesis se pueden resumir en las siguientes:

- Programa entero compacto para el BWC.
- Modelo de programación no lineal entera para el BWC (sólo factibilidad).
- BWC balanceado en tableros de ajedrez con caballos.
- Modelos para anti coloración con más de dos colores.
- Método por etapas.
- GRASP + calibración automática para OBWC.
- Algoritmo genético para OBWC.
- Calibración automática del Algoritmo genético.
- Función de aptitud y corrección para más de 3 colores.
- Propuesta multiobjetivo.

Adicional a las aportaciones antes mencionadas, se puede destacar que en este trabajo se intentó abordar el problema mediante un enfoque poliedral, primero para las familias de gráficas que se conoce están en P y posteriormente para algunas familias que se sospecha están en P como las bipartitas o los tableros de las reinas. En este enfoque no quedaba claro cómo generar los planos de corte que indujeran caras del poliedro pero es una línea de investigación que podría seguirse. En cuanto a la versión multiobjetivo del problema, se intentó trabajar el problema con las técnicas clásicas de optimización multiobjetivo de 1ra 2da y 3ra generación, sin éxito alguno. Se especula que es debido a que el frente de Pareto es discreto y no convexo.

Contenido

Lista de Figuras

Lista de Tablas

Lista de algoritmos

Capítulo 1
Preliminares

1.1. Teoría de gráficas

La teoría de gráficas es un campo de las matemáticas cuyos objetos principales de estudio son estructuras que constan de vértices y aristas que muestran alguna relación entre los vértices.

Definición 1.1

Una gráfica simple *es un par $G = (V, E)$ donde V es el conjunto de los vértices y E es un conjunto de aristas tal que cada elemento de E es un subconjunto con dos elementos de V. También se escriben como $V(G)$ y $E(G)$ respectivamente.*

Por ejemplo $V = \{1, 2, 3, 4, 5\}$ y $E = \{(1, 2); (2, 3); (3, 1); (5, 4); (5, 1); (3, 4); (4, 2)\}$, cuya representación se puede hacer de manera visual como muestra la Figura 1.1.

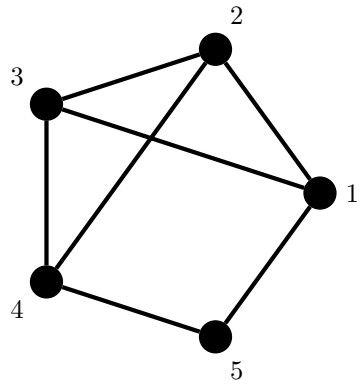

Figura 1.1: Ejemplo de una gráfica simple.

Definición 1.2

Una gráfica S es una subgráfica *de una gráfica G si cumple las condiciones $V(S) \subseteq V(G)$ y $E(S) \subseteq E(G)$.*

Definición 1.3

Una subgráfica inducida $< S >$ *de una gráfica* G *es una gráfica que cumple con* $V(S) \subseteq V(G)$ *y contiene todas las aristas incidentes en* $V(S)$.

Definición 1.4

Una subgráfica generadora *de* G *es una subgráfica inducida* $< S >$ *con* $V(G) = V(S)$.

Definición 1.5

Dos vértices u, v *son vecinos, si están unidos por una arista.*

Definición 1.6

Dado un vértice v *su vecindario son todos los vértices que son vecinos de* v *y se escribe* $N(v)$.

Definición 1.7

Dado un vértice v *su vecindario cerrado son todos los vértices que son vecinos de* v *incluyendo a* v *y se escribe* $N[v]$.

Definición 1.8

Dada una cierta propiedad en una gráfica simple, se dice que esta propiedad es maximal si no es posible mantenerla al añadir una arista.

Definición 1.9

Un clan es una subgráfica S *de* G, *tal que* S *es completa y maximal.*

Definición 1.10

Un paseo *en* G *es una sucesión alternada de vértices y aristas de* G *que inicia y termina en un vértice. Si el paseo termina en el mismo vértice donde inició se considera cerrado; si termina en uno diferente se considera abierto.*

Definición 1.11

Un camino *en* G *es un paseo en* G *donde todas las aristas son distintas. Si el camino es cerrado se le conoce como* circuito.

Definición 1.12

Una trayectoria *en* G *es un camino en* G *que no repite vértices, si es cerrado se le conoce como ciclo. Comúnmente un ciclo con* n *vértices se denota como* C_n.

Definición 1.13

Una gráfica es disconexa si para algún par de vértices $u, v \in V$ *no hay una trayectoria que los una.*

Definición 1.14

Un corte es un conjunto de aristas que al ser retirado de la gráfica, ésta queda disconexa.

Definición 1.15

Dada una gráfica $G = (V, E)$ si $V = A \cup B \cup C$ tal que A, B y C son disjuntos por pares y no hay aristas entre A y B, el conjunto separador de vértices es el subconjunto C.

Teorema 1.1

Dada una gráfica T los siguientes enunciados son equivalentes y definen a un árbol

1. *T es conexo y no tiene ciclos.*
2. *T no tiene ciclos y, si se añade alguna arista se forma un ciclo (acíclica maximal).*
3. *T es conexo y si se le quita alguna arista deja de ser conexo (conexo minimal).*
4. *T dos vértices cualquiera de G están conectados por una única trayectoria.*

Definición 1.16

Una gráfica cordal es una gráfica simple en donde cada ciclo de longitud 4 o más tiene una cuerda. Es decir no hay ciclos mayores a 4 sin aristas intermedias.

1.2. Complejidad computacional

La teoría de la complejidad computacional es un área de las ciencias de la computación que estudia el costo involucrado en la solución de los problemas. Se pretende medir la cantidad de recursos a utilizar, como por ejemplo: tiempo, espacio, etc. Una herramienta comúnmente usada en la complejidad computacional es el análisis de algoritmos que es la medición de los recursos usados por un algoritmo dado. Se desea analizar al menos dos aspectos:

- Cota superior: dar un buen algoritmo.
- Cota inferior: demostrar que ningún algoritmo es mejor.

La complejidad computacional además de estudiar el uso de recursos por los algoritmos, también estudia la complejidad de los problemas resueltos por dichos algoritmos, ello para conocer si el problema se puede resolver, después para saber si va a ser posible encontrar un buen algoritmo o no. Los tipos de problemas que por lo regular se encuentran son los siguientes:

- Problemas triviales.
- Problemas de decisión.

- Problemas de búsqueda.
- Problemas de conteo.
- Problemas de optimización.

En nuestro caso nos concentraremos en lo que son los problemas de decisión y los problemas de optimización.

Definición 1.17

Un problema P es una relación entre un conjunto I de instancias y un conjunto S de soluciones. Si $(i, s) \in P$ diremos que s es una solución para la instancia i.

Definición 1.18 (Problema de decisión)

Un problema de decisión *es aquel en el que las instancias se dividen en dos clases: sí y no.*

Definición 1.19 (Problema de optimización)

Un problema de optimización es aquel en el que se desea obtener las mejores soluciones de cada instancia. Cada instancia $i \in I$ incluye una función objetivo f_i de su conjunto de soluciones S_i a un conjunto con un orden total (usualmente los reales) y se desea obtener el menor valor posible de $f_i(s)$ con $s \in S_i$.

1.2.1. Algoritmo

Para responder a la pregunta ¿qué es un buen algoritmo? Primero se necesita responder, ¿qué es un algoritmo? De manera informal se puede ver que un algoritmo es un método efectivo expresado como una lista finita de instrucciones bien definidas. Comenzando desde un estado inicial y con una entrada (posiblemente vacía), las instrucciones describen un proceso determinista que, al ejecutarse, procederá a través de una cantidad finita de estados consecutivos, eventualmente generando una salida (posiblemente vacía) y terminando en un estado final.

Otra forma de definirlo de una manera formal es.

Definición 1.20

Un algoritmo A es una función de su conjunto de entradas I a su conjunto de salidas S. Si se ejecuta A con la entrada $e \in I$ y al terminar produce la salida $s \in S$, diremos que $A(e) = s$.

Además de esta definición los algoritmos que necesitaremos se deben asociar a la resolución de un problema dado.

Definición 1.21

Diremos que un algoritmo $A : I \to S$ resuelve un problema $P : I \to S$ si $A(e) = P(e)$ para

toda $e \in I$

De manera informal podemos decir que un *buen algoritmo*, es aquel que se ejecuta en un tiempo razonable (se puede realizar en un tiempo humanamente viable). Además se necesita poder expresar la cantidad de recursos utilizados por un algoritmo como una función, dichas funciones dan origen a lo que se conoce como clases de complejidad, por ejemplo:

1. Logarítmicas $\log_2 n$.
2. Lineales $3n + 5$.
3. Polinomiales $n^3 + 2n^2 + 3$.
4. Exponenciales 3^n.
5. Factoriales $n!$.

De aquí podemos decir que del 1 al 3 de la lista se consideran buenos algoritmos y el 4 y 5 son malos algoritmos en lo que respecta al uso de recursos.

1.2.2. Notación O, Ω y Θ

Para poder explicar esto con mayor precisión (sin depender de factores constantes, como el lenguaje de programación, la habilidad del programador o la potencia del equipo donde se ejecuten los algoritmos), se requiere de una medida de complejidad que mida el uso de los recursos en función del tamaño de su entrada. Para ello requeriremos de las siguientes notaciones que ignoran dichos factores.

Definición 1.22

Diremos que $f(n) \in O(g(n))$ *si existen constantes* $c > 0$ *y* $n_0 \geq 0$ *tales que* $f(n) \leq cg(n)$ *para toda* $n \geq n_0$.

Definición 1.23

Diremos que $f(n) \in \Omega(g(n))$ *si existen constantes* $c > 0$ *y* $n_0 \geq 0$ *tales que* $f(n) \geq cg(n)$ *para toda* $n \geq n_0$.

Definición 1.24

Diremos que $f(n) \in \Theta(g(n))$ *si* $f(n) \in O(g(n))$ *y* $f(n) \in \Omega(g(n))$.

Con estas notaciones observamos que se cumplen ciertas propiedades algebraicas.

Teorema 1.2 (Primer teorema de la suma)

Si $f_1(n) \in O(g_1(n))$ *y* $f_2(n) \in O(g_2(n))$ *entonces* $f_1(n) + f_2(n) \in O(g_1(n) + g_2(n))$.

Teorema 1.3 (Segundo teorema de la suma)

Si $f_1(n) \in O(g_1(n))$ *y* $f_2(n) \in O(g_2(n))$ *entonces* $f_1(n) + f_2(n) \in O(\text{máx}(g_1(n), g_2(n)))$.

Teorema 1.4 (Teorema del producto)

Si $f_1(n) \in O(g_1(n))$ *y* $f_2(n) \in O(g_2(n))$ *entonces* $f_1(n) \cdot f_2(n) \in O(g_1(n) \cdot g_2(n))$.

1.2.3. Problemas P y NP

Como ya se había mencionado antes, los algoritmos resuelven problemas. Entonces se tiene un algoritmo con cierto orden de ejecución y que resuelve un problema dado.

- Constante $O(1)$: determinar si un número es par.
- Logarítmico $O(\log n)$: búsqueda binaria.
- Lineal $O(n)$: suma de n bits.
- Polinomial $O(n^c)$: acoplamiento máximo.
- Exponencial $O(c^n)$: generación de cadenas c-arias.
- Factorial $O(n!)$: generación de permutaciones.

Los anteriores son los órdenes que corresponden a cierto algoritmo que resuelve el problema mencionado, sin embargo, eso no denota la complejidad del problema que resuelven. Para denotar la complejidad de los problemas son necesarios otros conceptos.

Se mencionó que las notaciones O, Ω y Θ son funciones del tamaño de la entrada, pero para que ello tenga sentido debemos mencionar cómo se representa dicha entrada. Una convención ampliamente utilizada es usar la codificación que requiera la menor cantidad de símbolos (bits) para representar a los objetos correspondientes.

Existen algoritmos que por sus características pueden clasificarse en las siguientes clases.

Definición 1.25

Un algoritmo se dice polinomial *si su tiempo de ejecución es* $O(p(n))$ *para algún polinomio* p *en el tamaño* n *de su entrada.*

Con respecto a los problemas que resuelven los algoritmos polinomiales tenemos la siguiente definición.

Definición 1.26

Un problema de decisión se dice polinomial *si existe algún algoritmo polinomial que lo resuelva. El conjunto de estos problemas es la clase* P.

Definición 1.27

Un algoritmo no determinista se dice polinomial *si su tiempo de ejecución es* $O(p(n))$ *para algún polinomio* p *en el tamaño* n *de su entrada.*

Definición 1.28

Un problema de decisión se dice polinomial no determinista *si existe algún algoritmo polinomial no determinista que lo resuelva. El conjunto de estos problemas es la clase NP.*

El término no determinista, está asociado a lo que se conoce como una máquina de Turing no determinista. Para entender las implicaciones que esto conlleva sólo se mostrará la diferencia entre una máquina de Turing determinista y la no determinista.

Definición 1.29

Una máquina de Turing T determinista acepta una entrada e si la ejecución de $T(e)$ lleva a T a un estado final $f \in F$, donde F es el conjunto de estados finales.

Definición 1.30

Una máquina de Turing T no determinista acepta una entrada e si alguna ejecución de $T(e)$ lleva a T a un estado final $f \in F$ donde F es el conjunto de estados finales.

La diferencia parece ser muy sutil, sin embargo, la máquina no determinista implica averiguar cual es la ejecución que implicó el estado final.

La división de los problemas en las clases anteriores nos lleva a plantearnos las siguientes clases para problemas de un tipo en particular.

Definición 1.31

Un problema de decisión D se dice NP-completo *si:*

- *Está en NP.*
- *Todo problema en NP se puede reducir a D en tiempo polinomial.*

Definición 1.32

Un problema D es NP-duro *si todo problema en NP se puede reducir a D en tiempo polinomial.*

Existen una gran variedad de problemas que caen en esta clase. Para profundizar más en el tema se recomienda leer [16].

Adicionalmente cabe mencionar que saber si $NP = P$ es un problema abierto, considerado uno de los Problemas del Milenio [25].

1.2.4. Problemas NP de optimización

Otra clase de problemas involucrada en este trabajo es la que tiene que ver con los problemas NP de optimización.

Definición 1.33

Un problema de NP optimización $OPT : I \to S$ cumple que:

1. *Su conjunto I de instancias se puede reconocer en tiempo polinomial (es decir, el problema de decisión $e \in I$ está en P).*
2. *Si $e \in I$ y $S(e)$ es su conjunto de soluciones entonces:*
   a) *Existe un polinomio p tal que si $s \in S(e)$ entonces $|s| \leq p(|e|)$.*
   b) *Si $|t| \leq p(|e|)$, se puede decidir en tiempo polinomial si $t \in S(e)$.*
3. *La función objetivo $m : I \times S \to \mathbb{Z}$ se puede calcular en tiempo polinomial.*

De lo anterior podemos definir la clase NPO para todos los problemas NP de optimización. Además se dice que si tenemos un problema $OPT \in NPO$ que se resuelve en tiempo polinomial. Entonces su versión de decisión D_{OPT} también se puede resolver en tiempo polinomial. En conclusión, si $P \neq NP$ y D_{OPT} es NP-completo, entonces OPT no se puede resolver en tiempo polinomial.

1.3. Programación matemática

Dentro del campo de estudio de la optimización existen una gran diversidad de herramientas que permiten la solución de problemas de optimización. En general existen dos formas de resolver un problema. La primera es mediante un algoritmo específico que resuelva dicho problema y la segunda es mediante una adaptación de una técnica general a la solución de nuestro problema. Quizás la más utilizada de estas técnicas es la programación matemática. En dicha técnica se parte de un modelo del problema propuesto. El modelo en su forma general consta de una o varias funciones objetivo y en ocasiones considerando algunas restricciones al modelo. Dependiendo de las características del modelo son las técnicas a utilizarse. La forma general de un programa matemático es la siguiente:

$$z = \max f(x) \tag{1.1}$$
$$g(x) \leq b \tag{1.2}$$
$$x \in D \tag{1.3}$$

De donde $f(x)$ es la función objetivo, $g(x)$ es el conjunto de restricciones, x el conjunto de variables de decisión y D el espacio de búsqueda de las variables, es decir los valores posibles que pueden tomar. Dependiendo de las características de todos esos elementos podemos tener los siguientes modelos.

Definición 1.34

Si $f(x)$ y $g(x)$ son lineales y el conjunto de variables x son continuas. El programa es **lineal**.

Definición 1.35

Si $f(x)$ y $g(x)$ son lineales y el conjunto de variables x son enteras. El programa es **entero**.

Definición 1.36

Si $f(x)$ y $g(x)$ son lineales y el conjunto de variables $x = x_c \cup x_d$, tal que x_c son las variables continuas y x_d son las variables discretas. El programa es **lineal entero mixto**.

Definición 1.37

Si $f(x)$ o $g(x)$ son no lineales y el conjunto de variables x son continuas. El programa es **no lineal**.

Definición 1.38

Si $f(x)$ o $g(x)$ son no lineales y el conjunto de variables x son enteras. El programa es **no lineal entero**.

Definición 1.39

Si $f(x)$ y $g(x)$ son no lineales y el conjunto de variables $x = x_c \cup x_d$, tal que x_c son las variables continuas y x_d son las variables enteras. El programa es **no lineal entero mixto**.

En cuanto a técnicas de solución para este tipo de modelos existe una amplia variedad de ellas resaltando las más comunes en la siguiente lista.

- Programación lineal (Método Simplex, Algoritmo de Karmarkar).
- Programación lineal entera o entera mixta (Ramificación y Acotamiento, Planos de corte).
- Programación no lineal (Métodos de newton, cuasi newton, mínimos cuadrados).
- Programación no lineal entera o no lineal entera mixta (Métodos de barrera).

Estos son solo algunos de los tipos de modelos así como de las técnicas existentes para resolverlos. Es cierto que existen otros modelos y también otro tipo de técnicas de solución pero no se profundizará en ello. Para mayor información sobre este tema se recomienda consultar [24]. En la actualidad existen herramientas computacionales libres y comerciales para resolver distintos modelos. Las mas destacadas son las siguientes.

- Gurobi (lineal,lineal entera, no lineal no lineal entera).
- lp-solve (lineal,lineal entera).
- GLPK (lineal,lineal entera).
- COIN-OR (lineal,lineal entera, no lineal no lineal entera).

Gurobi [19] es una herramienta comercial del estado del arte. Gurobi asegura ser la herramienta más avanzada en solución de programas matemáticos debido a que lo implementado

en ella consta de los más recientes descubrimientos del área. Existe una licencia ilimitada para uso académico.

lp-solve [9] y GLPK [9] son proyectos libres y mantenidos por la comunidad por lo que el acceso a características de descubrimientos recientes es limitada y lenta, no obstante, son gratuitos e incluso se puede modificar su código fuente.

COIN-OR [12] es una iniciativa apoyada por la Unión Europea y otras instituciones vinculadas a la Investigación de Operaciones dicha iniciativa consta en parte de un conjunto de herramientas para resolver problemas de programación matemática, estas herramientas son de acceso libre y tratan de estar al día en cuanto avances se refiere, aunque aún no son capaces de ganarle a Gurobi.

1.4. Métodos heurísticos

Cuando se habla de algoritmos, una característica de ellos es que para una entrada dada devuelven un resultado, además de que este resultado es el mismo siempre y es la respuesta correcta al problema estudiado. El problema resulta difícil puesto que es un problema NP-Duro ó NPO en el caso de los problemas de optimización, el obtener la respuesta correcta en un tiempo razonable se convierte en una situación compleja a menos que P=NP. En este momento es cuando se justifica el uso de métodos heurísticos.

Los métodos heurísticos son procedimientos de solución basados en reglas simples y que no requieren tanto conocimiento de la estructura subyacente del problema, en el caso de problemas de optimización lo que sacrifica es la garantía de optimalidad, dependiendo del problema existen situaciones en las que es la única opción viable para obtener una solución cercana al óptimo pero sin garantía de saberlo en algunas ocasiones. Existen distintos tipos de heurísticas y se clasifican según su alcance en Heurísticas ad-hoc y Meta-heurísticas. las primeras son procedimientos específicos para un problema dado, aprovechan alguna propiedad particular del problema estudiado para resolverlo. Por otro lado las Meta-heurísticas son procedimientos generales que se pueden adaptar en la solución de problemas de optimización, dicha adaptación depende de la habilidad y creatividad de quien resuelve el problema.

Las técnicas heurísticas también se clasifican en cuanto a la información que pueden brindar sobre la solución del problema. En ese caso se dice que hay heurísticas y algoritmos de aproximación. Las primeras no poseen información alguna sobre si la solución obtenida está cerca del óptimo, pero al mismo tiempo no requieren información sobre la estructura del problema. En el caso de los algoritmos de aproximación, ellos proveen una garantía de qué tan lejos quedaron en relación al óptimo aun sin conocer el óptimo. Para encontrar un algoritmo de aproximación primero se debe demostrar que el problema estudiado es aproximable y en segundo lugar, demostrar cual es la garantía de aproximación.

En años recientes se ha dado un crecimiento muy acelerado en cuanto al desarrollo de Meta-heurísticas y se ha hecho un esfuerzo por tratar de entender su funcionamiento, al mismo tiempo que se critica su uso indiscriminado para resolver cualquier problema en ocasiones sin averiguar si el problema o la instancia en particular se pudiesen resolver por algoritmos exactos.

En este trabajo la dificultad del problema tratado justifica el uso de técnicas heurísticas. En particular se abordará el uso de los Algoritmos Genéticos [21] así como la técnica conocida como Procedimientos de búsqueda miopes aleatorizados y adaptativos (GRASP, por sus siglas en inglés) [15].

1.4.1. GRASP

Definición 1.40

La calidad de una solución $s \in S$ es un mapeo $f : S \to \mathbb{R}$, donde S es el espacio de soluciones de un problema P.

El GRASP es un método Meta-heurístico propuesto por Feo y Resende en [15]. dicho procedimiento consta de dos etapas. Una etapa de construcción de la solución inicial y una etapa de búsqueda local. La etapa de construcción de la solución inicial se hace mediante un algoritmo glotón aleatorizado. Para entender en qué consiste primero hay que entender qué es un algoritmo glotón.

Algoritmo glotón

Un algoritmo glotón es aquel que en cada iteración toma la mejor decisión en ese momento, sin que ello implique el llegar al óptimo en muchos de los casos. Existen ejemplos de problemas que pueden resolverse de manera óptima con un algoritmo glotón como el problema de encontrar el árbol generador de peso mínimo. Pero también existen casos de problemas que no pueden ser resueltos de manera óptima con el glotón como por ejemplo el problema de la mochila o el problema del cambio de monedas.

A partir de esto, un algoritmo glotón aleatorizado no toma la mejor decisión en cada paso, construye una lista de decisiones candidatas, elige de manera aleatoria una y la agrega a la solución. Al determinar la lista de posibles decisiones candidatas surge la pregunta de qué tan grande debe ser la lista o hasta qué tanto permitimos que la decisión tomada no sea la mejor. Para ello la construcción de la lista se puede hacer bajo dos criterios.

- Tamaño de lista fijo.
- Tamaño de lista por valor.

Cuando el tamaño de la lista es fija se ordenan a las candidatas de mayor a menor en cuanto a la calidad de ser incluidas en la solución y solo se elige entre las k primeras.

Cuando el tamaño de la lista se define por valor se permite que una decisión candidata entre a la lista si su calidad queda dentro de un intervalo dado. Para un problema de maximización está dada por la siguiente expresión:

$$c_{min} + \alpha(c_{max} - c_{min}) \leq f(s \cup d_i) \leq c_{max}$$

Donde c_{min} es el valor más pequeño posible para cualquier decisión d_i, c_{max} es el valor más grande posible para cualquier decisión d_i y $\alpha \in [0, 1]$. Notar que si $\alpha = 0$, el algoritmo es totalmente aleatorio mientras que si $\alpha = 1$, el algoritmo es completamente glotón. Este parámetro se debe ajustar dependiendo del problema en particular.

Posterior a la construcción de la solución inicial se realiza un método de búsqueda local. En dicho método se establece un vecindario de la solución recién construida de tal manera que el método se pueda mover de la solución actual a la vecina por medio de un pequeño cambio. Se busca el mejor de los vecinos y si hay mejora con respecto a la solución actual, se cambia. El GRASP en forma general se muestra en el Algoritmo 1.1, mientras que la búsqueda local se puede ver en el Algoritmo 1.2.

Algoritmo 1.1 GRASP

Entrada: Un problema P, α, $maxI$.
Salida: Óptimo Local

```
1: s* = −∞
2: para i = 1 hasta maxI hacer
3:    s = ConstruccionGRASP(P, α)
4:    s' = BusquedaLocal(s)
5:    si f(s') < f(s) entonces
6:       s = s'
7:    fin si
8:    si f(s) < f(s*) entonces
9:       s* = s
10:   fin si
11: fin para
12: devolver s*
```

Claramente el algoritmo puede volverse multiarranque si se hace $maxI > 1$.

Algoritmo 1.2 Búsqueda Local

Entrada: s.
Salida: Óptimo local.

```
1: mientras ¬Condicion_de_Paro hacer
2:    s' = máx{s_c | s_c ∈ N(s)}
3:    si f(s') > f(s) entonces
4:       s = s'
5:    fin si
6: fin mientras
7: devolver s
```

1.4.2. Algoritmos genéticos

Los algoritmos genéticos fueron propuestos por Holland en [21] como un método metaheurístico que simula el proceso de selección natural propuesto por Darwin. Todo bajo un enfoque conocido como Neo Darwinista. En este proceso de selección natural se simulan los tres procesos más sobresalientes en la selección natural que son la **cruza**, la **mutación** y la **selección** de los más aptos, durante este proceso una solución a una instancia de un problema es vista como un individuo. Los individuos son evaluados por medio de una función de aptitud. Posteriormente y dependiendo del problema y de la forma en que dicho problema se adapte (codificación de las soluciones) a los algoritmos genéticos es la forma de funcionamiento de los operadores. En este trabajo nos centraremos en el caso en que las soluciones pueden ser representadas mediante cadenas binarias. Una vez aplicados los operadores se seleccionan los individuos que sobrevivirán para pasar a la siguiente generación y así sucesivamente hasta alcanzar un criterio de paro. Estos métodos se consideran poblacionales ya que trabajan con un conjunto de soluciones al que se denomina población.

Operadores de selección

- Selección uniforme: Se eligen los individuos sin importar su aptitud. Todos tienen la misma probabilidad de ser seleccionados.
- Selección por aptitud: Se evalúan los individuos y se elige al mejor evaluado.
- Selección por ruleta: Dependiendo del valor de la función de aptitud se establece una función de asignación de probabilidad proporcional a su valor de aptitud, haciendo que un individuo con un valor más alto sea más probable de ser elegido.
- Selección por torneo: Se seleccionan k individuos de la población con probabilidad uniforme o por ruleta y de esos k individuos se elige al mejor.
- Selección por muestreo determinista (dss): Se hace en dos etapas. En la primera etapa se calcula el valor esperado de cada individuo. Se genera población temporal con los individuos con valor esperado más alto y el resto de la población se llena con los individuos que tengan un valor más alto en la parte decimal de su valor esperado. En la segunda etapa se eligen los individuos de la población temporal de manera uniforme.
- Selección por muestreo estocástico residual (srs): Se hace en dos etapas. En la primera etapa se calcula el valor esperado de cada individuo. Se genera población temporal con los individuos con valor esperado más alto y el resto de la población se llena dándole una probabilidad de acuerdo a la parte fraccional. Por ejemplo para un individuo con un valor esperado de 1.4. se asegura una posición y se tiene 40 % de probabilidad de ser elegido de nuevo. En la segunda etapa se eligen los individuos de la población temporal de manera uniforme

El cálculo del valor esperado de un individuo s_i se hace con la siguiente ecuación

$$E(s_i) = |POP| \frac{f(s_i)}{\sum_{s_j \in POP} f(s_j)}$$

Donde POP es la población de individuos y f es la función de aptitud.

Operadores de cruza

Para la cruza tambien existen diversos operadores. los más comunes usados en cadenas binarias son los siguientes:

- Cruza de un punto: Se seleccionan dos individuos (padres) y una posición k entre 1 y $N-1$ en la cadena binaria. Los elementos del primer padre van al primer hijo desde la posición 0 hasta la posición $k-1$. El resto de posiciones de la cadena binaria del primer hijo se llenan con los elementos del segundo padre a partir de la posición k hasta la posición $N-1$. Si se desea un segundo hijo este se llena con las partes no utilizadas de las cadenas de los padres.

- Cruza multipunto: Se seleccionan dos individuos (padres) y un conjunto de p posiciones $\{k_1, \ldots k_p\}$, tal que $k_i \neq k_j$ para toda $i, j \in \{1, \ldots, p\},\ con\ i \neq j$ y las $k_i \in \{1 \ldots N-1\}$. Para el primer hijo se toma desde la posición 0 hasta la posición $k_1 - 1$ del primer padre luego desde k_1 hasta $k_2 - 1$ del segundo padre y así sucesivamente hasta completar la información del hijo. En caso de un segundo hijo se comienza obteniendo la información primero del segundo padre. Como lo muestra la Figura 1.2.

- Cruza uniforme: Se seleccionan dos individuos (padres) y la información del primer hijo se selecciona del primer padre o del segundo padre de manera aleatoria bit a bit. Para el segundo hijo se repite el proceso.

- Cruza par impar: Se seleccionan dos individuos (padres) y la información del primer hijo esta formada por los bits pares del primer padre y los bits impares del segundo. Mientras que el segundo hijo está formado por los bits impares del primer padre y los pares del segundo.

Operadores de mutación

La mutación en cadenas binarias se reduce a dos esquemas:

- Tomar un bit aleatorio de la cadena binaria y colocar su complemento.

- Para cada posición en la cadena binaria generar un número aleatorio q si $q \leq pm$, donde pm es la probabilidad de mutación, se cambia el bit por su complemento y si no, se ignora.

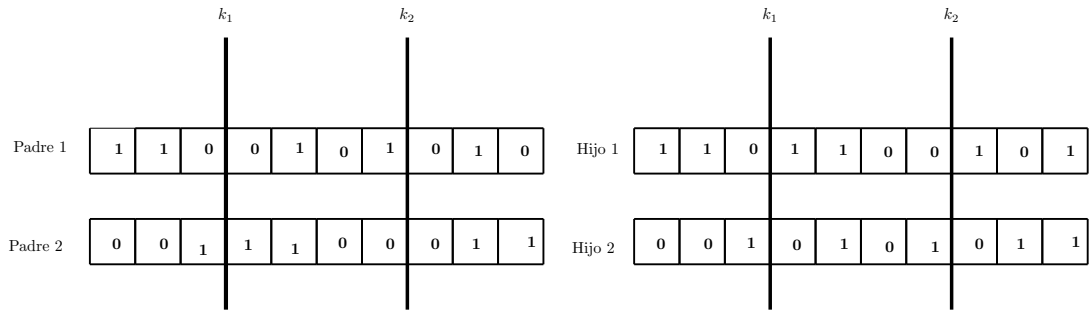

Figura 1.2: Ejemplo de cruza multipunto

1.4.3. Ajuste de parámetros

Un parámetro es algún valor de control necesario para la ejecución de algunas técnicas heurísticas. En ocasiones el seleccionar el valor adecuado cambia el comportamiento de la técnica ante el problema. Un ejemplo de parámetro es la probabilidad de mutación. Para algunos problemas un valor alto de mutación implica una intensificación en la búsqueda; sin embargo, en algunos casos un valor alto puede implicar una búsqueda alejada de la región factible.

Definición 1.41

Una configuración de parámetros c es un conjunto de parámetros para la ejecución de una meta-heurística H sobre un problema P.

Uno de los mayores inconvenientes al usar técnicas meta-heurísticas es que la gran mayoría poseen algunos parámetros a ser ajustados o calibrados. De no realizar este paso, el desempeño de la técnica podría ser muy malo o en ocasiones pudiera mejorarse más de lo que intuitivamente se espera. Existen varias formas de hacer este ajuste entre las que destacan:

- Prueba y error: Es un método bastante intuitivo y propenso a errores ya que la calibración se hace probando algunos parámetros e implementando en la ejecución los que a criterio del usuario fueron los "mejores".

- Búsqueda exhaustiva (grid search): En éste método se propone un conjunto de valores posibles para cada parámetro y se realiza la búsqueda de la mejor combinación de éstos. En parámetros discretos resulta sumamente efectiva pero no así en parámetros continuos, además de que el costo computacional es alto. El tiempo de ejecución de este algoritmo es de $O(P^d)$, donde P es la cantidad de parámetros y d es la máxima cantidad de valores posibles para uno de los parámetros.

- Búsqueda aleatoria: Se generan configuraciones aleatorias de parámetros respetando los rangos válidos para cada parámetro. Se verifica su desempeño hasta encontrar una que que sea aceptable para los requerimientos del problema.

- Técnicas de aprendizaje: Se basan en el uso de redes neuronales o máquinas de soporte vectorial para encontrar las mejores configuraciones, el único problema es que estas técnicas también requieren parámetros para ser calibrados.
- Co-evolución: Es utilizar un algoritmo que de manera paralela optimice los parámetros para el desempeño de la meta-heurística principal. El problema es que la segunda meta-heurística (heurística) requiere también un ajuste de parámetros.
- Autoadaptación: Toma a los parámetros como variables del problema a optimizar y les aplica el mismo proceso de la heurística utilizada salvo diferencias por los tipos de datos.
- Técnicas estadísticas. Similar a la versión de búsqueda aleatoria, se genera un conjunto de configuraciones de parámetros y se evaluan de manera secuencial pero se seleccionan las mejores bajo ciertos criterios de distribución de probabilidad.

En este trabajo se utilizó una técnica conocida como F-Race propuesta por Birattari en [10]. La técnica consiste de los siguientes pasos:

1. Se genera de manera aleatoria un conjunto inicial de configuraciones candidatas.
2. Se evalúan de manera secuencial las configuraciones candidatas con instancias de prueba.
3. Se descartan las peor evaluadas con evidencia estadística.
4. Se repiten 2 y 3 hasta encontrar una configuración ganadora o hasta que el tiempo de cómputo se agote.

Posteriormente en [23] se proponen algunas mejoras así como una herramienta computacional conocida como **irace** que facilita y automatiza el proceso de ajuste de parámetros.

1.5. Optimización multiobjetivo

Un problema de optimización multiobjetivo es aquel que contiene más de una función objetivo a optimizar y además donde los objetivos se contraponen. De manera general un problema de optimización multiobjetivo se puede ver de la siguiente manera:

$$z = \text{mín}(f_1(x), f_2(x), \ldots, f_k(x)) \tag{1.4}$$
$$x \in F \tag{1.5}$$

Donde las f_i son las funciones objetivo, x es el conjunto de variables de decisión y F es el espacio de soluciones factibles.

Un ejemplo de problema de optimización multiobjetivo es el de un portafolio de inversión ya que dado un instrumento de inversión, si se quieren mayores beneficios económicos el riesgo de

pérdida aumenta. Mientras que si se desea mantener el dinero seguro, el retorno de ganancias disminuye. Lo que buscaría el tomador de decisiones es una solución que minimice el riesgo y maximice las ganancias al mismo tiempo. El concepto de óptimo cambia en un problema múltiobjetivo y se introduce el concepto de óptimo de Pareto propuesto por economista Wilfredo Pareto. Para poder definir el concepto de óptimo de Pareto se requieren las siguientes definiciones.

Definición 1.42

Se dice que una solución $s_1 = (f_1(x_1), f_2(x_1), \ldots, f_k(x_1))$ **domina** *a una solución* $s_2 = (f_1(x_2), f_2(x_2), \ldots, f_k(x_2))$ *si* $f_i(x_1) \leq f_i(x_2)$ *para toda* $i \in \{1 \ldots k\}$ *y Existe una* $j \in \{1, \ldots k\}$ *tal que* $f_j(x_1) < f_j(x_2)$*. Se denota como* $s_1 \preceq s_2$*.*

Definición 1.43

Dos soluciones $s_1 = (f_1(x_1), f_2(x_1), \ldots, f_k(x_1))$ *y* $s_2 = (f_1(x_2), f_2(x_2), \ldots, f_k(x_2))$ *son* **no dominadas** *si existe una* $j \in \{1, \ldots, k\}$ *tal que* $f_j(x_1) > f_j(x_2)$*.*

Definición 1.44

Una solución s^* *es un* **óptimo de Pareto** *si no existe otra solución* s *que la domine.*

Definición 1.45

Al conjunto de óptimos de Pareto se le denomina frente de **Pareto***.*

En relación a la optimización multiobjetivo existen muchas técnicas para resolver este tipo de problemas, se puede encontrar un estudio a profundidad en el libro de Coello, *et al.* [11]. Aunque este último no abarca los más recientes avances en el área es una buena referencia sobre el tema. En éste trabajo se plantearán algunos modelos de optimización multiobjetivo para el problema de anticoloración. Cabe destacar que inicialmente se usaron las técnicas relativamente recientes de optimización multiobjetivo (NSGA-II, NSGA-III, SPEA2, SPEA3 y PAES) bajo las condiciones estándar, sin embargo, no tuvieron éxito.

Capítulo 2
Estado actual del conocimiento en anticoloraciones en gráficas

2.1. Problemas iniciales

Dos problemas significativos para la introducción del concepto de anticoloraciones de vértices en gráficas fueron los propuestos por Berge [8] y Lipton y Tarjan [22] en la octava década del siglo XX. En el problema propuesto por Berge se tiene un tablero de ajedrez de $n \times n$, b reinas negras y w reinas blancas, tales que $b + w \leq n$. ¿Es posible colocar todas las piezas en el tablero, de manera que ningún par de reinas blanca y negra se puedan atacar? Formalmente el problema se presenta como sigue:

Problema 2.1 (Reinas de dos colores)

Entrada: *Un tablero de ajedrez de $n \times n$, b y w enteros positivos.*
Salida: *1 si es posible colocar b reinas negras y w reinas blancas en el tablero, de manera que ningún par de reinas blanca y negra se puedan atacar, 0 en otro caso.*

Por otra parte, Lipton y Tarjan propusieron el problema conocido como Problema de separador de vértices (VSP por sus siglas en inglés). Dicho problema se presenta como sigue:

Problema 2.2 (VSP)

Entrada: *Una gráfica $G = (V, E)$ y $0 \leq \alpha \leq \frac{1}{2}$.*
Salida: *$A, B, C \neq \emptyset$ subconjuntos de V, tal que $A \cup B \cup C = V$, $A \cap B = B \cap C = A \cap C = \emptyset$, no hay aristas entre A y B, el tamaño de A o B es a lo más αn y $|C|$ es mínimo.*

El problema 2.2 sirvió para resolver una variante del problema 2.1, en la que se colocan en el tablero torres en lugar de reinas. Posteriormente, en 1997, Hansen *et al.*, [20] probaron que encontrar un separador mínimo para una gráfica es NP-Duro y describieron por primera vez una anticoloración de vértices con dos colores, pero sin mencionar el termino anticoloración. Es en 2005 cuando Berend *et al.*, [3] proponen el concepto de anticoloración, definiéndolo como una generalización del VSP.

Definición 2.1 (Anticoloración)

Una anticoloración de los vértices de una gráfica $G = (V, E)$ es una coloración parcial de los vértices $a : V \to S \cup I$, siendo S el conjunto de colores e $I = \{incoloro\}$, tal que para cada $\{v, w\} \in E$, $a(v) = a(w)$ o al menos uno de los dos vértices esta en I

Dicha definición nos permite introducir el Problema de Anticoloracion de Gráficas (PAC), cuya definición es la siguiente:

Problema 2.3 (PAC)

Entrada: *Una gráfica $G = (V, E)$ y $c_1, c_2, \ldots, c_k$ enteros positivos tal que $\sum_{i=1}^{k} c_i \leq |V|$.*
Salida: *1 si se pueden colorear c_1 vértices de color 1, c_2 vértices de color 2, ..., c_k vértices de color k, bajo la condición de anticoloración, 0 en otro caso*

Posteriormente, en 2007, Berend *et al.*, [4] demuestran que el PAC es NP-Completo para $k = 2$ reduciéndolo al VSP. Esta versión de dos colores se conoce como el problema de coloración blanco y negro (BWC por sus siglas en inglés), cuya resolución ha sido abordada desde el ámbito de los algoritmos exactos para algunas familias de gráficas y mediante metaheurísticas para el caso general. La definición formal del BWC es la siguiente:

Problema 2.4 (BWC)

Entrada: *Una gráfica $G = (V, E)$ y b y w enteros positivos .*
Salida: *1 si se pueden colorear b vértices del color negro y w vértices del color blanco en $V(G)$ bajo la condición de anticoloración, 0 en otro caso.*

2.2. Gráficas del movimiento de piezas en tableros de ajedrez

El Problema 2.1 se puede modelar como un problema de teoría de gráficas, en el cual se representan con vértices las celdas del tablero de ajedrez y con aristas los desplazamientos que se pueden realizar en un solo movimiento de la pieza. De esta forma, al variar los movimientos posibles para cada tipo de pieza, variarán en forma equivalente las condiciones de conexión de los vértices por medio de las aristas. Para cada tipo de pieza de ajedrez $t \in \{rey, reina, torre, caballo, alfil\}$ se define la gráfica $G_t = (V_t, E_t)$ como sigue:

1. Por cada posición en el tablero de ajedrez se coloca un vértice en V_t.

2. Existe una arista $\{u, v\}$ entre dos vértices $u, v \in V_t$ si es posible desplazarse desde la posición asociada al vértice u a la posición asociada al vértice v y viceversa, mediante un movimiento válido de la pieza de ajedrez tipo t.

Por lo tanto, resolver el Problema 2.4 sobre la gráfica G_t de las reinas es equivalente a resolver el Problema 2.1, puesto que para cada par de vértices adyacentes en G_t, por la anticoloración los dos deben ser del mismo color o al menos uno de ellos incoloro, mientras que en el tablero la adyacencia representa dos posiciones en donde si hay piezas con distinto color, estas pueden atacarse.

En la Figura 2.1 se puede observar la gráfica asociada a los movimientos del caballo en un tablero de 4×4, en ella cada vértice tiene su casilla correspondiente como etiqueta. Por otro lado en la Figura 2.2 se puede observar el efecto de colocar un caballo en la posición indicada.

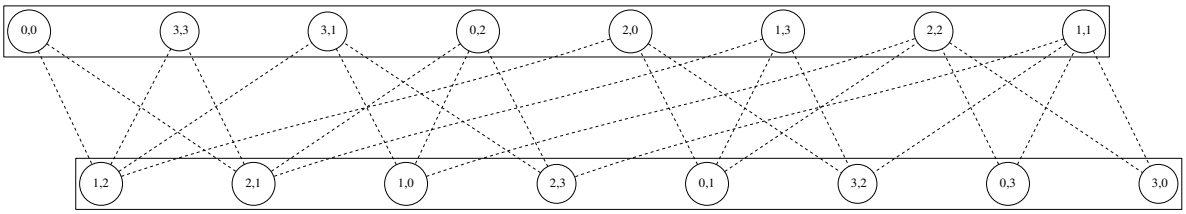

Figura 2.1: Gráfica de los movimientos del caballo en un tablero de 4×4.

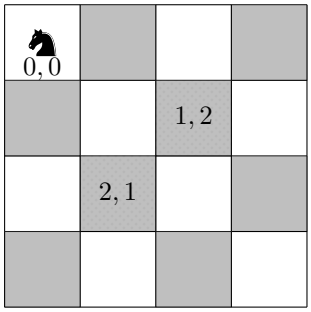

Figura 2.2: Tablero de 4×4 con un caballo negro.

Note que colocar un caballo negro en la posición $(0, 0)$ del tablero es equivalente a colorear de negro el vértice con etiqueta $0, 0$ de su gráfica de movimientos. Al mismo tiempo las casillas que quedan cubiertos por el caballo colocado, se convierten en vértices que solo pueden ser negros o incoloros en la gráfica de movimientos.

Casi todas las aportaciones en este problema provienen de Berend, *et al.*, y se pueden resumir en las siguientes:

- Un algoritmo para la construcción de soluciones en gráficas planas [3].
- Fórmulas y formas de anticolorear tableros de ajedrez para los reyes, torres, alfiles y conjeturas para el caso de los caballos y las reinas [7].

- Un algoritmo para la anticoloración de árboles [7].
- Un algoritmo para la anticoloración de gráficas cordales [26].
- Un algoritmo de Búsqueda Tabú para gráficas generales [6].
- Una mejora en cuanto a la obtención de la solución del caso de las torres [2].

2.3. BWC en gráficas planas

Antes de continuar con la descripción del estado del arte es necesario describir la versión de optimización de BWC (OBWC por sus siglas en inglés). En el OBWC en vez de decidir si b y w vértices pueden ser o no pintados de blanco y negro respectivamente, se fija el número de vértices negros y se busca colorear el conjunto que permita colorear el mayor número de vértices de color blanco bajo la condición de anticoloración. Formalmente se define como en el Problema 2.5.

Problema 2.5 (OBWC)

Entrada: *Una gráfica $G = (V, E)$ y $b \leq n$ un entero positivo*
Salida: *Un conjunto $B \subseteq V$ con $|B| = b$ y un conjunto W de cardinalidad máxima bajo la condición de anticoloración.*

En [3] se obtiene como resultado principal un algoritmo de tiempo $O(n)$ para gráficas planas. Dicho resultado se puede ver en el Teorema 2.2. Este algoritmo está a su vez basado en un resultado de Lipton y Tarjan [22] que se puede observar en el Teorema 2.1.

Teorema 2.1 (Lipton y Trajan (1979))

Dada una gráfica plana $G = (V, E)$ con $n = |V|$ vértices. G se puede partir en tres conjuntos, T_1, T_2 y C tal que no hay aristas entre T_1 y T_2. Ni T_1 ni T_2 contienen más de $\frac{2}{3}n$ vértices y C no contiene más de $2\sqrt{2}\sqrt{n}$ vértices.

Teorema 2.2 (Berend, et al., (2010))

Dada una gráfica plana $G = (V, E)$ con $n = |V|$ vértices y $b \leq n$ un entero positivo. El algoritmo 2.1 encuentra una anticoloración con b vértices negros y $w \geq n - b - O(\text{mín}\{\sqrt{n}, b\})$ y su tiempo de ejecución es $O(n)$.

Cabe destacar que tanto el El Teorema 2.2 como el Algoritmo 2.1 no funcionan cuando $b \geq n - 6\sqrt{2}(1 + \sqrt{2/3})\sqrt{n}$ o dicho de otro modo, no es posible encontrar una anticoloración con al menos un vértice blanco aunque dicha anticoloración exista. En el mismo artículo, donde se mencionan los resultados sobre gráficas planas, se proponen algunas mejoras como un posible trabajo futuro, en cuanto al desempeño del algoritmo pero no en cuanto a la obtención de la solución. En este punto es importante destacar que el Algoritmo 2.1 no da

Algoritmo 2.1 Algoritmo para OBWC en gráficas planas

Entrada: $G = (V, E)$ plana, $b \leq n$ un entero positivo

Salida: Una anticoloración con b vértices negros y $w \geq n - b - O(\text{mín}\{\sqrt{n}, b\})$ vértices blancos

1: $G' = G,\ n' = |V|,\ b' = b,\ S, T_1, T_2 = \emptyset$.
2: **mientras** $\sqrt{n'} \leq b \leq 6\sqrt{2}(1 + \sqrt{2/3})\sqrt{n'}$ **hacer**
3: $(T_1, T_2, n_1, n_2) = LiptonTarjan(G')$
4: $b_1 = n_1 > b'$
5: $b_2 = n_2 > b'$
6: **selección** (b_1,b_2)
7: **caso** (1,1)**:**
8: **si** $\frac{2}{3}n_1 \leq b'$ **entonces**
9: $G' =< T_1 >$
10: $n' = n_1$
11: **si no**
12: $G' =< T_2 >$
13: $n' = n_2$
14: **fin si**
15: **caso** (1,0)**:**
16: $S = S \cup T_2$
17: $b' = b' - n_2$
18: $G' =< T_1 >$
19: $n' = n_1$
20: **caso** (0,0)**:**
21: $S = S \cup T_1$
22: $b' = b' - n_1$
23: $G' =< T_2 >$
24: $n' = n_2$
25: **fin selección**
26: **fin mientras**
27: **si** $b' < \sqrt{n'}$ **entonces**
28: Elegir cualquier conjunto de b' vértices con el grado más pequeño y agregarlos a S
29: **si no**
30: Colorear de blanco los vértices de grado más pequeño, siempre y cuando el número de estos vértices y sus vecinos sea menor que $n' - b'$
31: **fin si**
32: **devolver** *soluciones*

una garantía de optimalidad, sólo puede asegurarnos encontrar una solución factible para gráficas planas en un tiempo razonablemente corto siempre que el valor de b esté dentro del límite antes mencionado. El caso de las gráficas planas es sumamente interesante ya que se pudo observar que las gráficas planas son las más sencillas de resolver tanto para los solvers de programación entera como para las heurísticas de tipo glotón.

2.4. BWC en tableros de ajedrez

Los tableros de ajedrez son el entorno de donde surgió el BWC. El caso de los alfiles es un caso trivial ya que se puede ocupar todo el tablero colocando los alfiles blancos en las casillas blancas y los alfiles negros en las casillas negras sin preocuparse porque se ataquen. En el caso de las torres Lipton y Tarjan responden a la pregunta en el caso de tableros de $n \times n$. Pero es hasta 2015 donde Berend, *et al.*, proponen una solución y un algoritmo de tiempo lineal para el caso de tableros rectangulares [2]. En el mismo trabajo proponen una mejora del tiempo de ejecución del algoritmo a sublineal basándose en un enfoque poliedral. La propuesta de solución del caso de las torres está dada por el programa no lineal entero (2.1–2.3).

$$\begin{aligned} z &= \text{máx}(m-r)(n-c) && (2.1)\\ rc &\geq b && (2.2)\\ r, c &\in \mathbb{Z}^+ \; con \; r \in [1, m] \; y \; c \in [1, n] && (2.3) \end{aligned}$$

Donde r y c denotan el número de filas y columnas respectivamente, ocupadas por las torres negras.

El programa no lineal entero (2.1–2.3) pretende encontrar el rectángulo rc que maximice el área del rectángulo azul como se muestra en la Figura 2.3. Se sabe que en el óptimo $c = \lceil \frac{b}{r} \rceil$ y sin pérdida de generalidad se puede asumir que existe una solución óptima tal que $r \leq c$ por lo que $r \leq \lceil \frac{b}{r} \rceil$. Lo que modifica el programa no lineal entero (2.1–2.3), como se muestra a continuación.

$$\begin{aligned} z &= \text{máx}(m-r)(n-\lceil \frac{b}{r} \rceil) && (2.4)\\ r &\in \mathbb{Z}^+ \; con \; \frac{b}{n} \leq r \leq \text{mín}(m, \lceil \sqrt{(}b) \rceil) && (2.5) \end{aligned}$$

Con el programa no lineal entero (2.4–2.5), se puede implementar fácilmente un algoritmo de tiempo de ejecución $O(m)$. Basta con explorar todos los valores posibles de r. En [2] también se demuestra que el valor de r no puede obtenerse mediante una búsqueda binaria debido a que la función objetivo (ecuación 2.4) no es monótona. Así que en el artículo previamente citado se menciona una manera de lograrlo mediante la relajación no lineal del programa no lineal entero (2.4–2.5).

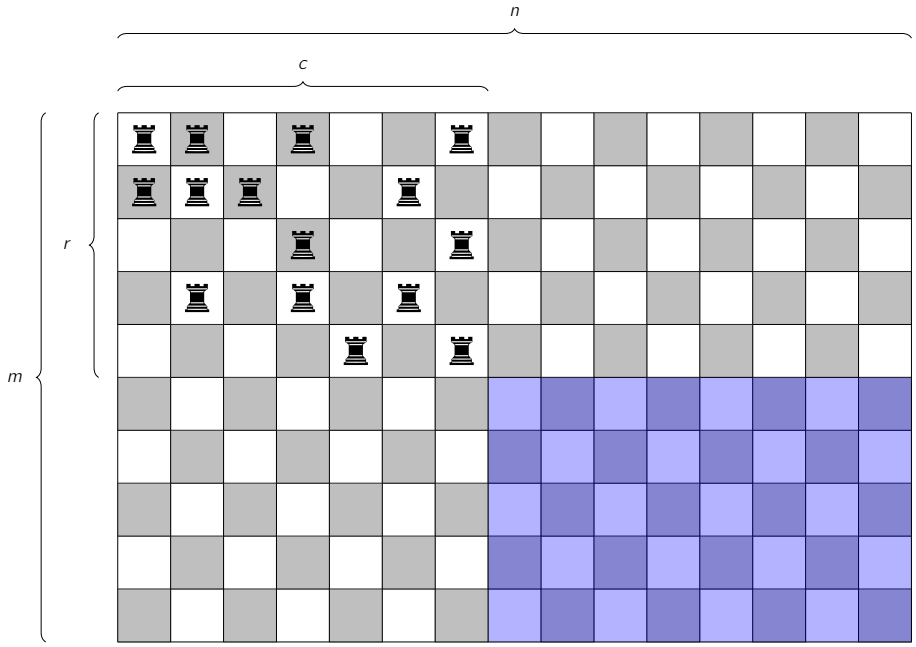

Figura 2.3: Ejemplo de una posible colocación de las torres negras

En el caso de más de tres colores también se prueba que la forma de acomodo en el tablero se puede extender a más de dos colores siempre y cuando se cumpla que $c_1 + \ldots + c_{k-j} \leq \frac{mn}{4}$ con k clases de color y $j = 0, 1, \ldots, k-1$.

2.4.1. BWC y el caso de los reyes

El caso de los reyes fue abordado en [5]. En la referencia se demuestran fórmulas y formas de colocar los reyes que dependen completamente del tamaño de b en relación con el tamaño del tablero. En este caso. Berend, *et al.*, dan fórmulas para el caso de tableros rectangulares y tableros toroidales.

Los resultados se pueden resumir en los Teoremas 2.3 y 2.4.

Teorema 2.3 (Berend, et al., (2008))

Dado un tablero de ajedrez **toroidal** *de $m \times n$ con $m \leq n$ y sea b el número de reyes de color negro a ser colocados en el tablero. El número óptimo de reyes de color blanco que se pueden colocar en el tablero es:*

$$w_{opt} = mn - b - \begin{cases} 2\lceil 2\sqrt{b} \rceil + 4 & Si\ 1 \leq b \leq \left(\frac{m}{2} - 1\right) \\ 2m + s & Si\ \ Si\ \left(\frac{m}{2} - 1\right) < b \leq mn - \left(\frac{m}{2} + 1\right)^2 \\ 2\lceil \sqrt{4(mn-b) - 2} \rceil - 4 - R & Si\ mn - \left(\frac{m}{2} + 1\right)^2 < b \leq mn - 9 \\ mn - b & Si\ mn - 9 < b \leq mn \end{cases} \tag{2.6}$$

$$\textit{donde } s = \begin{cases} 0 & \textit{Si } b \cong 0 (mod\ m) \\ 1 & \textit{Si } b \cong -1 (mod\ m) \\ 2 & \textit{Si } b \ncong 0, -1 (mod\ m) \end{cases} \qquad R = \lfloor \tfrac{r}{t-1} \rfloor \ r = b-(a-1)\lceil b/a \rceil \ y \ a = \lceil \sqrt{b} \rceil$$

La forma en la que se deben colocar los reyes en el tablero se puede observar en la Figura 2.4

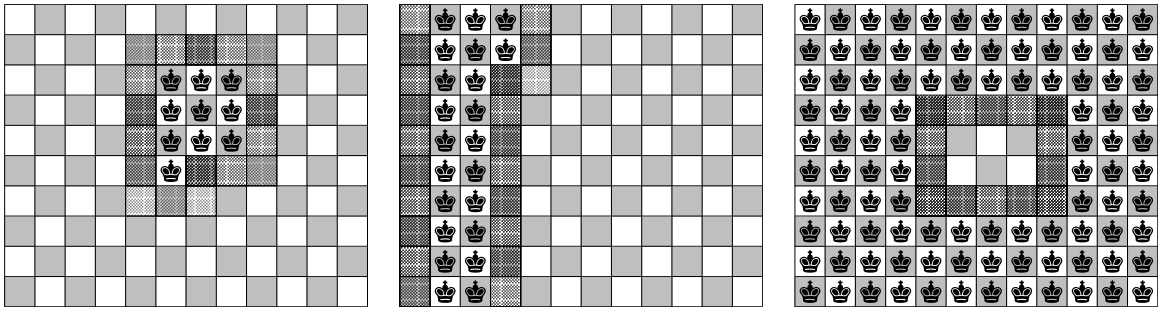

Figura 2.4: b pequeña (izquierda), b intermedia (centro), b grande (derecha).

En el caso de tableros planos (no toroidales), los resultados se enuncian en el Teorema 2.4. La forma en la que se colocan los reyes es la misma, salvo en el tamaño medio que hay un ligero cambio debido a que ahora hay casillas que no son vecinas con respecto a como lo eran en el toro. Dicha configuración se puede ver en la Figura 2.5.

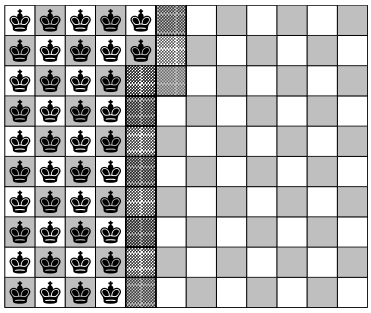

Figura 2.5: Tablero **no toroidal** de reyes con una b intermedia.

Teorema 2.4 (Berend, et al., (2008))

Dado un tablero de ajedrez **no toroidal** *de $m \times n$ con $m \leq n$ y sea b el número de reyes de color negro a ser colocados en el tablero. El número óptimo de reyes de color blanco*

que se pueden colocar en el tablero es:

$$w_{opt} = mn - b - \begin{cases} 2\lceil\sqrt{b}\rceil + 1 & Si\ 1 \leq b \leq \left(\frac{m-1}{2}\right)^2 \\ m + s' & Si\ \ Si\ \left(\frac{m-1}{2}\right)^2 < b \leq mn - \left(\frac{m+1}{2}\right)^2 \\ \lceil\sqrt{4(mn-b)-2}\rceil - 1 & Si\ mn - \left(\frac{m+1}{2}\right)^2 < b \leq mn - 4 \\ mn - b & Si\ mn - 4 < b \leq mn \end{cases} \quad (2.7)$$

$$donde\ s' = \begin{cases} 0 & Si\ b \cong 0 (mod\ m) \\ 1 & Si\ b \ncong 0 (mod\ m) \end{cases}$$

2.4.2. El caso de los caballos y las reinas

Estos son los casos menos explorados. En el caso de los caballos no se sabe como se comporta w_{opt} en función de b. En [5] se conjeturó algo sobre un caso especial sobre los caballos del cual hablaremos en la siguiente sección y lo mismo sucede con el caso de las reinas.

2.4.3. El caso balanceado

Comenzaremos definiendo el concepto de balance. Cabe destacar que en este caso el balance utilizado se conoce como balance generalizado y dicho balance no implica necesariamente igualdad. En el balance generalizado se pretende encontrar el más grande de todos los mínimos sujetos a una cierta propiedad. Por ejemplo en el caso de las anticoloraciones se busca encontrar el máximo del mínimo de las cardinalidades de las clases de color para cualquier tupla factible de cardinalidades.

Definición 2.2 (Balance generalizado)

Un balance generalizado se define como:

$$z = \text{máx}\{\text{mín}\{c_1, c_2, \ldots, c_k\}\}$$

con $c_i \in \mathbb{Z}^+$ *para toda* $i \in \{1, 2, \ldots, k\}$

A partir de la Definición 2.2 es posible definir el PAC-Balanceado como se muestra en el Problema 2.6

Problema 2.6 (PAC-Balanceado)

Entrada: *Una gráfica* $G = (V, E)$ *y un entero* k *que denota el número de clases de color*
Salida: $\text{máx}\{\text{mín}\{c_1, c_2 \ldots c_k\}\}$ *bajo la condición de anticoloración.*

Si hacemos $k = 2$ se tiene BWC-Balanceado. En el caso particular del BWC-Balanceado en tableros de ajedrez se define como en el Problema 2.7.

Problema 2.7 (BWC-Balanceado en tableros de ajedrez)

Entrada: *Un tablero de ajedrez de $m \times n$, la pieza del tipo $t \in \{rey, reina, caballo, torre, alfil\}$.*
Salida: $z = \max\{\min\{b, w\}\}$*, bajo la condición de anticoloración.*

En [5] Berend, *et al.*, mencionan algunos resultados sobre el caso balanceado en tableros no toroidales de $n \times n$. Dichos resultados están resumidos en la Tabla 2.1.

Caso	Resultado	Estado
Reyes	$\frac{n^2-n}{2}$, n impar $\frac{n^2}{2} - n$, n par	Demostrado
Alfiles	$\frac{n^2-1}{2}$, n impar $\frac{n^2-3n}{2}$, n par	Demostrado (Trivial)
Caballos	$\frac{n^2-2n-1}{2}$, n impar $\frac{n^2}{2} - n$, n par	Conjeturado (Falso)
Torres	$\frac{n^2-1}{4}$, n impar $\frac{n^2}{4}$, n par	Demostrado
Reinas	$0.132n^2$, n impar	Conjetura (sobre la forma de acomodo)

Tabla 2.1: Resultados sobre BWC-Balanceado en tableros de ajedrez.

Uno de los logros de esta tesis es la demostración del caso de tableros de $m \times n$ en el caso de los reyes y la demostración del caso rectangular para los caballos. Se mostró de manera experimental que una conjetura de [5] el caso de n impar era incorrecta por medio de un contraejemplo. Posteriormente mediante el Teorema 3.1. En la Figura 2.6 se muestra la solución de Berend *et al.*, y nuestra propuesta. Por otro lado en la Figura 2.7 se muestra la conjetura de la forma de acomodo en el caso de las reinas.

2.5. BWC en árboles y gráficas cordales

El caso de los árboles está basado en los resultados de Lipton y Tarjan sobre el VSP. En este caso Berend, *et al.*, proponen un algoritmo de programación dinámica sobre árboles que hace uso de los separadores generados por el VSP sobre árboles. A diferencia del caso de las gráficas planas, en este caso sí se garantiza la optimalidad del resultado. Los resultados principales se pueden resumir en el Teorema 2.5 y el Algoritmo 2.2.

Problema 2.8 (BWC en Árboles)

Entrada: *Un árbol enraizado T de n vértices*
Salida: *Un arreglo max_w, el cual contiene para cada $0 \leq b \leq n$, la máxima w tal que exista una anticoloración de T.*

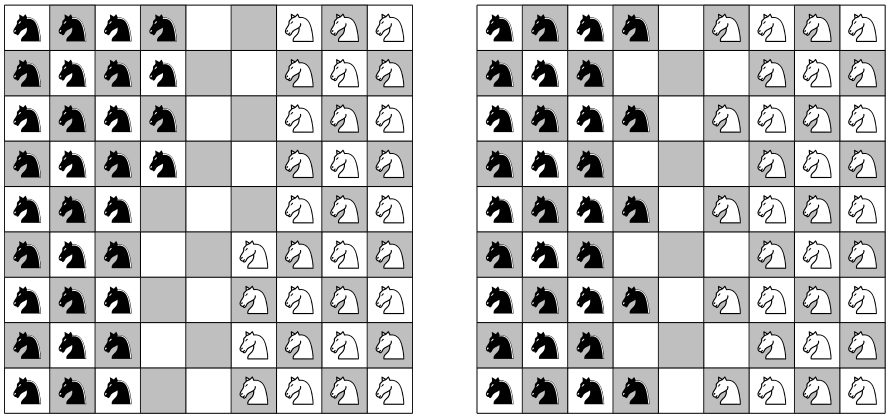

Figura 2.6: Caso incorrecto de la conjetura de Berend *et al.*, (izquierda), nuestra propuesta (derecha).

Teorema 2.5 (Berend, et al., (2009))

El Problema 2.8 se puede resolver por el Algoritmo 2.2 en tiempo $O(n^2 \log^3 n)$

Algoritmo 2.2 Algoritmo para encontrar una BWC para un árbol con raíz r

Entrada: Un árbol T de n vértices y una raíz r

Salida: Un arreglo max_w que contiene la máxima w para una $0 \leq b \leq n$

1: **para cada** hoja $v \in T$ **hacer**
2: $v.max_w[negro] = [0]$
3: $v.max_w[blanco] = [1]$
4: $v.max_w[incoloro] = [0]$
5: **fin para**
6: $r.max_w = generarTabla(T)$
7: **para** $i = 0$ **hasta** n **hacer**
8: $max_w[i] = \text{máx}\{r.max_w[negro][i], r.max_w[blanco][i], r.max_w[incoloro][i]\}$
9: **fin para**
10: **devolver** max_w

En el Algoritmo 2.2 en la linea 6 se menciona un algoritmo para generar una tabla a partir de T y que dicha tabla se hace con respecto a la raíz r. Dicho procedimiento es el algoritmo de programación dinámica que asigna colores de manera recursiva dependiendo del color que se asigne en las hojas del árbol. El Algoritmo 2.2 calcula la mejor opción en cuanto a color que se le asigne al padre dados los colores de los hijos y lo lleva hasta el nodo raíz.

El caso de las gráficas cordales es presentado en [26]. Este caso se basa primordialmente en las siguientes definiciones:

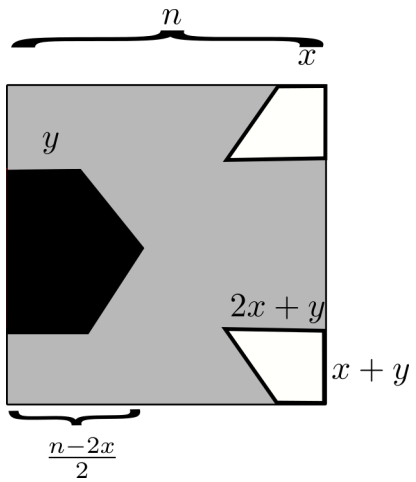

Figura 2.7: Propuesta de Berend *et al.*, sobre la forma de acomodo de las reinas.

Definición 2.3

Sea $G = (V, E)$ una gráfica cordal. La gráfica de intersección de clanes denotada por $C(G) = (V_c, E_c, \mu)$ con $\mu : E_c \to \mathbb{N}$ y está dada por:

1. *El conjunto de vértices V_c es el conjunto $\{K_1, K_2, \ldots, K_m\}$ de los clanes maximales de G.*
2. *El conjunto de aristas $E_c = \{(K_i, K_j) | K_i, K_j \in V_c \text{ y } K_i \cap K_j \neq \emptyset\}$.*
3. *$\mu(K_i, K_j) = |K_i \cap K_j|$ con $(K_i, K_j) \in E_c$.*

Definición 2.4

Sea G una gráfica cordal y $C(G)$ su gráfica de intersección de clanes. Un árbol de clanes de G es el árbol generador de peso máximo de $C(G)$

Por otro lado. El Teorema 2.6 caracteriza a las gráficas cordales a partir de sus clanes y menciona que los separadores mínimos son un clan.

Teorema 2.6 (Berend, et al., (2012))

Una gráfica G es cordal si y sólo si cada separador mínimo de G es un clan.

El algoritmo presentado en [26] se basa en el hecho probado en el Teorema 2.6 para desarrollar un algoritmo que obtiene todas parejas (b,w) que corresponden a anticoloraciones no dominadas o dicho de otro modo, aquellas anticoloraciones donde el conjunto de vértices incoloros es de tamaño mínimo. El algoritmo se ejecuta en tiempo $O(n^4)$.

2.6. BWC en gráficas generales

En [6] se propone la búsqueda tabú como método para resolver el OBWC en gráficas generales. La búsqueda tabú es un método general de solución de problemas de optimización propuesto por Fred Glover en [17] y [18], que tuvo su mayor auge en la década de los 90s del siglo pasado. A grandes rasgos la búsqueda tabú es una técnica de búsqueda local que bloquea temporalmente algunas regiones del espacio de búsqueda para evitar caer en óptimos locales. El procedimiento general de una búsqueda tabú se puede ver en el Algoritmo 2.3

Algoritmo 2.3 Búsqueda tabú

Entrada: P un problema de optimización y A_c un conjunto de parámetros del algoritmo
Salida: s_{mejor} un óptimo local

1: $s_0 = generar\_solucion\_inicial(P)$
2: $s_{mejor} = s_0$, $ListaTabu = ListaTabu \cup s_0$
3: **mientras** !Criterio_de_paro **hacer**
4: $s' = b\_local(s_{mejor}, N(s_{mejor}))$
5: **para cada** $s_n \in N(s_{mejor})$ **hacer**
6: **si** $buscar(ListaTabu, s_n) = 0$ **&** $eval(s_n) > eval(s')$ **entonces**
7: $s' = s_n$
8: **fin si**
9: **fin para**
10: **si** $eval(s') > eval(s_{mejor})$ **entonces**
11: $s_{mejor} = s'$
12: **fin si**
13: $ListaTabu = ListaTabu \cup s'$
14: **si** $num\_elementos(ListaTabu) > TamanoMax$ **entonces**
15: $Eliminar\_1er\_elemento(ListaTabu)$
16: **fin si**
17: **fin mientras**
18: **devolver** s_{mejor}

La búsqueda tabú es un mecanismo de búsqueda local que pretende evitar la convergencia prematura a un óptimo local, mediante el acotamiento del espacio de búsqueda por medio de la prohibición de movimientos hacia vecindarios que pudieron ser ya explorados. La prohibición no es permanente por lo que si después se requiere explorar alguna zona en particular, el algoritmo lo permite.

La forma en la que se pueden evitar temporalmente explorar partes del espacio de búsqueda es mediante una lista conocida como lista tabú que evita un movimiento especifico, en el caso del BWC una solución vecina se da sobre una gráfica con vértices con color negro asignado, en la solución vecina se toma un vértice negro y se vuelve incoloro y se toma un incoloro y se vuelve negro. La búsqueda tabú almacena qué vértices cambiaron de color y les prohíbe

volver a cambiar.

La propuesta de Berend *et al.*, menciona que un tamaño de lista tabú de 10 es lo mejor según experimentos aunque nunca detalla en qué consistieron dichos experimentos. Para realizar la búsqueda en el vecindario se le permite al algoritmo la exploración de $C \cdot |V|$ vecinos de un conjunto de $O(|V|^2)$ fijando el valor de $C = 20$. otra vez se hace mención que la C se obtiene de manera empírica pero no se detalla cómo.

En el caso del OBWC, la solución inicial se genera de dos maneras diferentes:

1. Aleatoria.

2. Pseudo glotona.

Puesto que en el OBWC el número de vértices negros es fijo, basta con asignar a b vértices el color negro de manera aleatoria y contar el número de vértices blancos posibles bajo esa asignación. En el caso de la inicialización pseudo glotona. Se asigna color negro vértice por vértice. Cada que se agrega un nuevo vértice se crea una lista de tamaño dado que contiene los vértices que al ser asignados maximizan la cantidad de vértices que pueden tener el color blanco asignado y se elige de esa lista un elemento de manera aleatoria y así sucesivamente hasta tener b vértices asignados. Después de ello, Berend, *et al.*, contabilizan el número de vecinos afectados por esta asignación y hacen la siguiente operación.

$$w = n - b - vecinos\_afectados$$

Posteriormente se realiza una búsqueda local asignándole color negro a un vértice incoloro y quitandole color a uno negro previamente asignado. Esto último para no afectar el número de negros que es fijo.

Aunque en [6] se presentan otros algoritmos como una búsqueda local con reinicio, un recocido simulado y un GRASP, sólo resulta ser competitivo el caso del GRASP. En este trabajo se ha desarrollado un algoritmo genético que es competitivo con la búsqueda tabú, adicionalmente se realiza una calibración del GRASP para mejorar los resultados originales. Puesto que en [6] reportan sólo los mejores resultados encontrados para un conjunto de instancias. Nuestro punto de comparación es alcanzar la misma solución o mejorarla.

En [6] se proponen un par de modelos de programación matemática. Un programa entero (2.8–2.12) y un programa no lineal (2.13–2.18). En este trabajo se proponen dos nuevos modelos. Uno que permite una implementación más sencilla de las meta-heurísticas y otro que permite ver el BWC como un problema multiobjetivo y llevarlo a más de dos colores.

Dada una gráfica $G = (V, E)$, se tiene que:

$$x_v = \begin{cases} 1 & \text{Si al vértice } v \text{ se le asigna color } \textbf{negro}. \\ 0 & \text{En otro caso.} \end{cases}$$

$$y_v = \begin{cases} 1 & \text{Si al vértice } v \text{ se le asigna color } \textbf{blanco}. \\ 0 & \text{En otro caso.} \end{cases}$$

$$
\begin{aligned}
z &= \text{máx} \sum_{v \in V} y_v && (2.8)\\
\sum_{v \in V} x_v &= b && (2.9)\\
x_v + y_v &\leq 1 \ \forall \ v \in V && (2.10)\\
x_u + y_v &\leq 1 \ \forall \ \{u, v\} \in E && (2.11)\\
x_v, y_v &\in \{0, 1\} \ \forall \ v \in V && (2.12)
\end{aligned}
$$

La función objetivo (2.8) maximiza el número de vértices que se colorean de blanco. La restricción (2.9) fija el número de vértices que deben tener color negro asignado. El conjunto de restricciones (2.10) permite que un vértice tenga a lo más un color asignado y el conjunto de restricciones (2.11) asegura que en cada arista sólo uno de los vértices terminales tiene un color, es decir si uno de los terminales es negro, el otro no puede ser blanco y viceversa.

$$
\begin{aligned}
z &= \text{máx} \sum_{v \in V} \left(x_v^2 + y_v^2\right) && (2.13)\\
\sum_{v \in V} x_v &= b && (2.14)\\
\sum_{v \in V} y_v &= w && (2.15)\\
x_v + y_v &\leq 1 \ \forall \ v \in V && (2.16)\\
x_u + y_v &\leq 1 \ \forall \ \{u, v\} \in E && (2.17)\\
0 \leq x_v, y_v &\leq 1 \ \forall \ v \in V && (2.18)
\end{aligned}
$$

En este caso el modelo es cuadrático puesto que la función objetivo (2.13) alcanza el máximo cuando x_v o y_v es 1 para algún v (esta es una propuesta de Berend *et al.*, no demostrada) incluye en este caso un número fijo de vértices de color blanco (2.15). Cabe destacar que el modelo cuadrático (2.13–2.18) es sólo de factibilidad por lo que se puede inferir que la función objetivo maximiza la factibilidad.

Capítulo 3
Anticoloraciones en gráficas y algoritmos exactos

Previamente se habló de todas las aportaciones que se han hecho de anticoloraciones. Como se pudo constatar este tema es bastante amplio y complejo, además de que ha sido poco abordado. En el caso concreto de este trabajo podemos resumir las contribuciones en algoritmos exactos a las siguientes:

- Programa entero compacto para el BWC.
- Modelo de programación no lineal entera para el BWC (sólo factibilidad).
- BWC balanceado en tableros de ajedrez con caballos.
- Modelos para anti coloración con más de dos colores.
- Método por etapas.

3.1. Programa entero compacto para el BWC

Una de las contribuciones de [6] es un modelo de programación entera en este modelo la función objetivo maximiza el número de vértices blancos, tiene una restricción que fija el número de vértices negros (b), $|V|$ restricciones para asegurar que un mismo vértice no tenga más de un color asignado y $2|E|$ restricciones, para asegurar que dada una arista con vértices u, v, en dicha arista sólo uno de sus vértices terminales tenga a lo más uno de ambos colores. Si sumamos todas las restricciones que tienen a x_v con v fijo incluyendo la restricción con la que un vértice no puede tener asignado dos colores (sin incluir la restricción de un número fijo de vértices negros), tenemos el siguiente programa entero:

$$x_v = \begin{cases} 1 & \text{si al vértice } v \text{ se le asigna color } \textbf{negro} \\ 0 & \text{en otro caso.} \end{cases}$$

$$y_v = \begin{cases} 1 & \text{si al vértice } v \text{ se le asigna color } \textbf{blanco} \\ 0 & \text{en otro caso.} \end{cases}$$

$$z = \max \sum_{v \in V} y_v \tag{3.1}$$

$$\sum_{v \in V} x_v = b \tag{3.2}$$

$$\sum_{s \in N[v]} y_s \leq |N[v]|(1 - x_v) \quad \text{para toda } v \in V \tag{3.3}$$

$$x_v, y_v \in \{0, 1\} \tag{3.4}$$

donde $N[v]$ es el vecindario cerrado de v, nótese que esta versión es equivalente a la propuesta por Berend, *et al.*, pero es más fácil de programar debido al tamaño del archivo a utilizar ya que tenemos una restricción por vértice, es decir que este programa tiene $|V| + 1$ restricciones en contraste con el modelo original que tiene $O(|E|)$ restricciones. Un ejemplo de la reducción se puede ver en el Ejemplo 3.1. Note que en este caso el conjunto de restricciones (3.3) se interpreta como que si un vértice v tiene asignado el color negro, entonces ni ese vértice v ni sus vecinos pueden ser de color blanco, en caso de que el vértice v no tenga asignado el color negro, entonces ese vértice como sus vecinos pueden ser de color blanco.

Ejemplo 3.1

Dada la gráfica $G = (V, E)$ de la Figura 3.1 el modelo de programación entera basado en el programa entero (2.8–2.12) correspondiente es:

$$
\begin{aligned}
z &= \text{máx } y_1 + y_2 + y_3 + y_4 && (3.5)\\
x_1 + x_2 + x_3 + x_4 &= b && (3.6)\\
x_1 + y_1 &\leq 1 && (3.7)\\
x_1 + y_2 &\leq 1 && (3.8)\\
x_1 + y_3 &\leq 1 && (3.9)\\
x_1 + y_4 &\leq 1 && (3.10)\\
x_2 + y_3 &\leq 1 && (3.11)\\
x_2 + y_1 &\leq 1 && (3.12)\\
x_2 + y_3 &\leq 1 && (3.13)\\
x_2 + y_4 &\leq 1 && (3.14)\\
x_3 + y_3 &\leq 1 && (3.15)\\
x_3 + y_1 &\leq 1 && (3.16)\\
x_3 + y_2 &\leq 1 && (3.17)\\
x_4 + y_4 &\leq 1 && (3.18)\\
x_4 + y_1 &\leq 1 && (3.19)\\
x_4 + y_2 &\leq 1 && (3.20)\\
x_i, y_i &\in \{0, 1\}, i = 1, 2, 3, 4. && (3.21)
\end{aligned}
$$

Si sumamos de la restricción 3.7 a la 3.10 correspondientes al vértice 1, obtenemos la siguiente expresión

$$4x_1 + y_1 + y_2 + y_3 + y_4 \leq 4$$

Reacomodando y factorizando obtenemos

$$y_1 + y_2 + y_3 + y_4 \leq 4(1 - x_1)$$

Que corresponde a una restricción del modelo compacto para el vértice 1. Se repite el mismo procedimiento con cada vértice.

La reducción del número de restricciones no implica una mejora en cuanto al tiempo de ejecución, puesto que es el mismo modelo pero con una representación más compacta. Note que en la desigualdad 3.3 si al vértice en cuestión se le asigna color negro, el número de vértices que pueden ser blancos en su vecindario es 0. Por el contrario si el vértice en cuestión no es negro, entonces tanto dicho vértice como su vecindario pueden ser blancos.

El programa entero compacto es el que se usó en este trabajo tanto en la solución exacta como para desarrollar las heurísticas del capítulo siguiente.

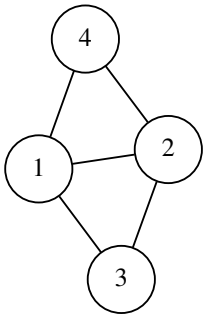

Figura 3.1: Gráfica del Ejemplo 3.1

3.2. Modelo de programación no lineal entera para el BWC

Recordemos que el problema de anticoloración está pensado como un problema de decisión, por lo que también proponemos el siguiente programa no lineal entero de factibilidad, cuya función objetivo es la parte no lineal. En este caso las variables de decisión están definidas como en el programa entero compacto. Dados b y w dos números enteros positivos.

$$z = \text{mín} \sum_{\{u,v\}\in E} x_u y_v + \sum_{v\in V} x_v y_v \tag{3.22}$$

$$\sum_{v\in V} x_v = b, \tag{3.23}$$

$$\sum_{v\in V} y_v = w \tag{3.24}$$

$$x_v, y_v \in \{0,1\} \tag{3.25}$$

El programa entero (3.22–3.25) nos sirve principalmente para medir la factibilidad, ya que minimiza el número de aristas bi-coloreadas y el que ningún vértice tenga asignado más de un color, por lo que se desea que el valor de la función objetivo dé un valor de 0. Si el programa (3.22–3.25) no tiene una solución con $z = 0$ tiene dos posibles causas:

- Un vértice tiene asignado mas de un color.
- No es posible colocar b vértices negros y w vértices blancos, sin que existan aristas con terminales bicoloreadas.

La primera situación se puede evitar de dos formas. Una es penalizar esa parte de la función objetivo, es decir que la función objetivo quede

$$z = \text{mín} \sum_{\{u,v\}\in E} x_u y_v + \lambda(\sum_{v\in V} x_v y_v)$$

donde λ es un valor positivo de penalización. La segunda manera de evitarlo es regresar a la forma lineal de esa parte para evitar la infactibilidad es decir cada término $x_v y_v$ de $\sum_{v\in V} x_v y_v$ se convierte en restricciones de la forma $x_v + y_v \leq 1$ para toda $v \in V$. En el segundo caso está claro que, si es posible poner una anticoloración factible con b y w fijos. Entonces $z = 0$, en particular la parte de la función objetivo relacionada con las aristas, es decir $\sum_{\{u,v\}\in E} x_u y_v$. Note que si los terminales de una arista $\{u, v\}$ están bicoloreados entonces $x_u y_v = 1$, lo cual implica un incremento en la función objetivo y no así con aristas monocromáticas o incoloras.

1. Si $x_v y_v = 1$ para todo $v \in V$ es una solución sin sentido (un mismo vértice es blanco y negro a la vez) por lo tanto en la función objetivo $\sum_{v\in V} x_v y_v$ tiene mayor prioridad.

2. Si $\sum_{\{u,v\}\in E} x_u y_v > 0$ en el óptimo, no es posible generar una anticoloración con b vértices negros y w vértices blancos.

En este caso al ser un modelo cuadrático requiere de algoritmos que permitan resolver programas no lineales enteros. En la literatura se recurre al uso de métodos de barrera para la solución dichos métodos se encuentran disponibles a través de Gurobi.

En el caso de que una gráfica $G = (V, E)$ y dos números enteros positivos b, w no tengan una anticoloración entonces se dice que tiene una partición mínima. Donde dicha partición consta de aristas bicoloreadas. Una manera de forzar a que esto suceda es si $b + w = n$ debido a que se quita la posibilidad de la existencia de incoloros. Esto nos lleva al problema de partición de gráficas que se define como sigue:

Problema 3.1 (Problema de partición de gráficas (PPG))

Entrada: *Una gráfica $G = (V, E)$, una función de costos $w : E \to \mathbb{Q}$ y un entero k.*
Salida: *k subconjuntos de vértices tal que, $V_1 \cup V_2 \cup \ldots \cup V_k = V$, $V_i \cap V_j = \emptyset$ para $i, j = 1, 2, \ldots k$, $i \neq j$ y la suma de los costos de las aristas que unen a los conjuntos sea mínima.*

El Problema 3.1 está pensado para más de dos conjuntos, y se busca minimizar un costo. Si hacemos $k = 2$ y $w(e) = 1$ para toda $e \in E$ entonces estamos frente al caso especial que es equivalente a la solución del programa entero no lineal anterior. El problema es conocido como el problema de corte mínimo para el cual ya existen algoritmos de tiempo polinomial que lo resuelven aún para costos no unitarios, cabe destacar que esto sólo funciona cuando no es importante el tamaño de los subconjuntos en los que se divide la gráfica. Si deseamos que ambos conjuntos sean del mismo tamaño, entonces se sabe que el problema es NP-Duro aún con costos unitarios [16] y es conocido como el Problema de Bisección de Gráficas (PBG).

En cuanto al PPG y en particular al PBG, existe un estudio mucho más a profundidad que se puede revisar en [14].

En la Fig. 3.2 se puede ver como hay 4 aristas que unen a ambos conjuntos.

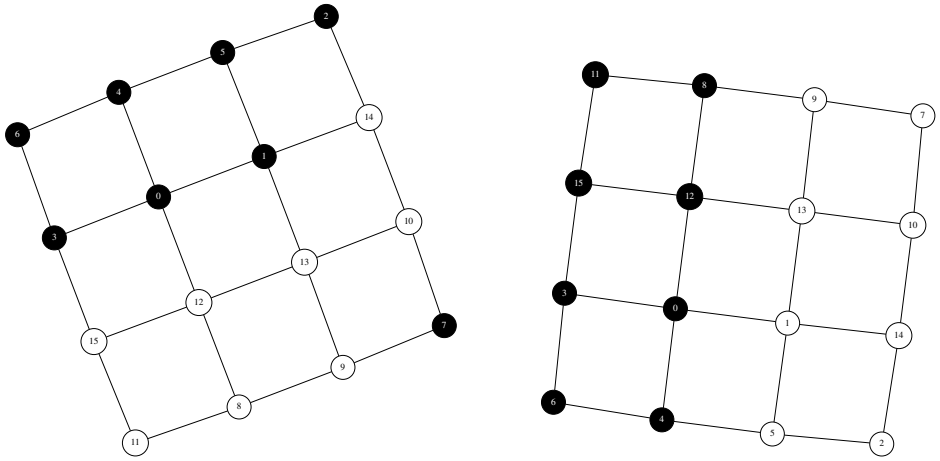

Figura 3.2: Ejemplo de dos posibles particiones de una gráfica en dos subconjuntos del mismo tamaño con diferentes costos

Como se puede observar, dado que $n = b+w$, no existe la posibilidad de que haya vértices incoloros. Se puede decir que encontrar una bisección de una gráfica es equivalente a minimizar la infactibilidad de una anticoloración con $b = w$ y $b + w = n$. Dado que la definición de balance para PBG es una definición donde los tamaños de los conjuntos deben ser $\frac{|V|}{2}$ implica que $|V|$ debe ser de tamaño par.

3.2.1. PBG y balance generalizado

Recordemos que en la definición de balance generalizado se busca encontrar máx{mín$\{b, w\}$}, lo cual no implica necesariamente que $b = w$. Esta versión de balance nos permite tratar problemas de balanceo donde $b \neq w$. La forma inicial de definir el problema de bisección de gráficas con balanceo generalizado (PBG-balanceado) queda como sigue:

Problema 3.2 (PBG Balanceado)

Entrada: *Una gráfica* $G = (V, E)$.
Salida: $z =$ máx{mín$\{b, w\}$} *con b y w las cardinalidades de los conjuntos en los que se particionó la gráfica con el mínimo número de aristas que conecten ambos conjuntos.*

Siguiendo la definición del Problema 3.2. El programa no lineal entero quedará como un programa multi-objetivo en donde uno de los objetivos es encontrar los valores de b y w que den un balance y por el otro lado encontrar el número mínimo de aristas que unen a ambos

conjuntos de la partición, lo cual nos genera el programa no lineal entero multi-objetivo (3.26 – 3.31).

$$
\begin{aligned}
Z &= \left(\text{máx}\{\text{mín}\{b, w\}\}, \text{mín} \sum_{\{u,v\}\in E} x_u y_v \right) && (3.26)\\
\sum_{v\in V} x_v &= b && (3.27)\\
\sum_{v\in V} y_v &= w && (3.28)\\
b + w &= |V| && (3.29)\\
x_v + y_v &\leq 1 \ \forall \ v \in V && (3.30)\\
x_v, y_v &\in \{0, 1\}, \ b, w \in \mathbb{Z}^+ && (3.31)
\end{aligned}
$$

Para poder tratar este problema es necesario hacer unas cuantas modificaciones en la función objetivo con lo que el programa (3.26 – 3.31) se convertirá en el programa (3.32 – 3.39).

$$
\begin{aligned}
Z &= \left(\text{máx}\, \theta, \text{mín} \sum_{\{u,v\}\in E} x_u y_v \right) && (3.32)\\
\theta &\leq b && (3.33)\\
\theta &\leq w && (3.34)\\
\sum_{v\in V} x_v &= b && (3.35)\\
\sum_{v\in V} y_v &= w && (3.36)\\
b + w &= |V| && (3.37)\\
x_v + y_v &\leq 1 \ \forall \ v \in V && (3.38)\\
x_v, y_v &\in \{0, 1\}. \ b, w \geq 0 && (3.39)
\end{aligned}
$$

El programa (3.32 – 3.39) se convierte en un programa mono-objetivo por medio de restricciones ε. En este método una de las funciones objetivo se vuelve una restriccion colocándole como cota superior a ε que es un parámetro para minimizar la otra función objetivo como se muestra en el programa (3.40 – 3.48)

$$
\begin{aligned}
Z &= \max \theta && (3.40)\\
\theta &\leq b && (3.41)\\
\theta &\leq w && (3.42)\\
\sum_{v \in V} x_v &= b && (3.43)\\
\sum_{v \in V} y_v &= w && (3.44)\\
x_v + y_v &\leq 1 \ \forall \ v \in V && (3.45)\\
b + w &= |V| && (3.46)\\
\sum_{\{u,v\} \in E} x_u y_v &\leq \varepsilon && (3.47)\\
x_v, y_v &\in \{0,1\}.\ b, w \geq 0 && (3.48)\\
& && (3.49)
\end{aligned}
$$

Debido a que el problema es discreto se puede obtener una descripción discreta del frente de Pareto partiendo de un valor de ε y reducirlo de uno en uno hasta que el problema se vuelva infactible. Está claro que ε no puede ser cero o estaríamos frente a una anticoloración. El Algoritmo 3.1 muestra el procedimiento a seguir.

Algoritmo 3.1 Método para optimización bi-objetivo vía restricciones ε

Entrada: $G = (V, E)$, valor inicial ε_0
Salida: m soluciones del PBG-balanceado

1: $\varepsilon = \varepsilon_0$
2: **mientras** $\varepsilon \geq 0$ **hacer**
3: S=resolverPNLE(programa((3.40 – 3.48),ε))
4: **si** $S_{Factible} = 1$ **entonces**
5: $soluciones = soluciones \cup S$
6: **fin si**
7: $\varepsilon = \varepsilon - 1$
8: **fin mientras**
9: **devolver** m *soluciones*

3.3. BWC en tableros de ajedrez

Como se mencionó en el capítulo anterior. Hay dos casos que son de interés. El primero es el caso balanceado para los reyes (BWC-Balenceado-reyes) en tableros rectangulares ya

que sólo se demostró para tableros cuadrados y el caso balanceado para los caballos (BWC-Balanceado-caballos). Puesto que la conjetura mostrada en [5] resultó parcialmente verdadera debido a un detalle en los tableros impares.

3.3.1. El caso balanceado con reyes en tableros de $m \times n$

Para el caso del BWC-Balanceado-reyes partiremos del Teorema 2.4. Existen 2 principales problemas en la demostración. En primer lugar ¿Qué se pretende buscar? y en segundo ¿Cómo usar el Teorema 2.4? Para poder definir lo que se está buscando nos auxiliaremos de una versión para tableros de ajedrez del programa entero (3.1–3.4) pero cuya función objetivo sea máx{mín$\{b, w\}$}. El programa entero se define en (3.50–3.54). Dado un tablero de $m \times n$. Para cada posición (i, j) en el tablero se tiene.

$$x_{ij} = \begin{cases} 1 & \text{si la pieza negra es colocada en la posición i,j} \\ 0 & \text{en otro caso} \end{cases}$$

$$y_{ij} = \begin{cases} 1 & \text{si la pieza blanca es colocada en la posición i,j} \\ 0 & \text{en otro caso} \end{cases}$$

$$\text{máx}\{ \quad \text{mín} \quad \{b, w\}\} \tag{3.50}$$

$$\sum_{i=1}^{m}\sum_{j=1}^{n} x_{i,j} \quad = \quad b \tag{3.51}$$

$$\sum_{i=1}^{m}\sum_{j=1}^{n} y_{i,j} \quad = \quad w \tag{3.52}$$

$$\sum_{s,r \in N[i,j]} y_{s,r} \quad \leq \quad |N[i,j]|(1 - x_{i,j}) \quad i = 1, \ldots, m, \ j = 1, \ldots, n \tag{3.53}$$

$$x_{i,j}, y_{i,j} \quad \in \quad \{0, 1\} \tag{3.54}$$

Donde $N[i, j]$ son las casillas que se pueden alcanzar desde la casilla (i, j) en un movimiento incluyendo la casilla (i, j). Denotemos con $\phi_t(m, n)$ a la solución óptima que se obtiene al resolver el programa entero (3.50–3.54) para la pieza t. En este primer caso $t = rey$. El corolario que se desprendería del Teorema 2.4 queda como sigue.

Corolario 3.1

Dado un tablero de ajedrez de $m \times n$ con $1 \leq m \leq n$, sean b, w el número de reyes negros y blancos respectivamente, entonces los posibles valores de ϕ_{rey} son los siguientes:

$$\textit{Caso 1: } \phi_{rey}(m, n) = \frac{m(n-1)}{2} - 1 \quad \textit{si } m \textit{ par, } n \textit{ par}$$

$$\textit{Caso 2: } \phi_{rey}(m, n) = \frac{m(n-1) - 1}{2} \quad \textit{si } m \textit{ impar, } n \textit{ par}$$

Caso 3: $\phi_{rey}(m,n) = \dfrac{m(n-1)}{2}$ *si m indistinta, n impar*

Para el caso balanceado, la forma de utilizar el Teorema 2.4 es primero afirmar que el número óptimo de reyes blancos dados $b = \phi_{rey}(m,n)$ reyes negros, corresponde con el caso de un tablero de tamaño medio como lo muestra el Teorema 2.4. Es decir la formula es la descrita en la ecuación 3.55.

$$w_{opt} = mn - b - (m + s') \text{ Si } \left(\frac{m-1}{2}\right)^2 < b \leq mn - \left(\frac{m+1}{2}\right)^2 \tag{3.55}$$

donde

$$s' = \begin{cases} 0 & \text{Si } b \cong 0 (mod\ m) \\ 1 & \text{Si } b \ncong 0 (mod\ m) \end{cases}$$

Para corroborar la afirmación previamente hecha. Enunciamos y demostramos el Lema 3.1

Lema 3.1

Sea $m \leq n$ con m, n enteros positivos y $b = \phi_{rey}(m,n)$ el número de reyes negros, entonces para el caso balanceado

$$\left(\frac{m-1}{2}\right)^2 < b \leq mn - \left(\frac{m+1}{2}\right)^2$$

Demostración. La demostración sólo se hace para el Caso 1 del Corolario 3.1 pero es análoga en todos los demás casos. Si $m \leq n$ entonces.

$$\left(\frac{m+1}{2}\right)^2 \leq m\left(\frac{m+1}{2}\right)^2 \leq n\left(\frac{m+1}{2}\right)^2$$

además

$$mn - \left(\frac{m+1}{2}\right)^2 \geq mn - m\left(\frac{m+1}{2}\right)^2 \geq mn - n\left(\frac{m+1}{2}\right)^2$$

con lo que

$$mn - \left(\frac{m+1}{2}\right)^2 \geq mn - n\left(\frac{m+1}{2}\right)^2$$

y

$$mn - \left(\frac{m+1}{2}\right)^2 \geq mn - n\left(\frac{m+1}{2}\right) \geq mn - n\left(\frac{m+1}{2}\right)^2$$

y por último

$$mn - n\left(\frac{m+1}{2}\right) \geq \frac{m(n-1)}{2} - 1 = b$$

Por otro lado

$$\left(\frac{m-1}{2}\right)^2 \leq m\left(\frac{m-1}{2}\right) \leq m\left(\frac{n-1}{2}\right)$$

Por lo tanto

$$\left(\frac{m-1}{2}\right)^2 < m\left(\frac{n-1}{2}\right)$$

y se puede corroborar fácilmente que

$$\left(\frac{m-1}{2}\right)^2 < m\left(\frac{n-1}{2}\right) - 1 = b$$

■

A partir del Lema 3.1 ahora es posible utilizar la ecuación 3.55.

Demostración. (Corolario 3.1). Se demostrará solo el caso 3 debido a que es el que muestra de manera más clara el comportamiento de s', los casos son análogos a éste. Dado $b = \frac{m(n-1)}{2}$, se tiene que

$$w_{opt} = mn - \frac{m(n-1)}{2} - (m + s') = \frac{m(n-1)}{2} - s'$$

de donde claramente

$$\text{mín}\left\{\frac{m(n-1)}{2}, \frac{m(n-1)}{2} - s'\right\} = \frac{m(n-1)}{2} - s'$$

para corroborar que es máximo, tomemos $b + k$ y w^k_{opt} su respectiva solución óptima, ahora bien,

$$w^k_{opt} = \frac{m(n-1)}{2} - (k + s')$$

de aquí podemos decir que

$$\text{mín}\left\{\frac{m(n-1)}{2} + k, \frac{m(n-1)}{2} - (s' + k)\right\} = \frac{m(n-1)}{2} - (s' + k)$$

y por lo tanto

$$\text{máx}\left\{\frac{m(n-1)}{2} - s', \frac{m(n-1)}{2} - (s' + k)\right\} = \frac{m(n-1)}{2} - s'.$$

■

De la anterior demostración sólo resta saber cual es el valor de s'. Sabemos que se define como

$$s' = \begin{cases} 0 & \text{Si } b \cong 0(mod\ m) \\ 1 & \text{Si } b \ncong 0(mod\ m) \end{cases}$$

O dicho de otro modo $s' = 0$. Si $m|b$ (m divide exactamente a b), es decir

$$\frac{\frac{m(n-1)}{2}}{m} = p$$

con $p \in \mathbb{Z}$. desarrollando:

$$\frac{\frac{m(n-1)}{2}}{m} = \frac{m(n-1)}{2m} = \frac{n-1}{2}$$

Como n es impar entonces $n = 2q + 1$ con $q \in \mathbb{Z}$ con lo que $\frac{2q}{2}$ es entero y por tanto $s' = 0$ para el Caso 3. En la Figura 3.3, se pueden ver los 3 casos del Corolario 3.1.

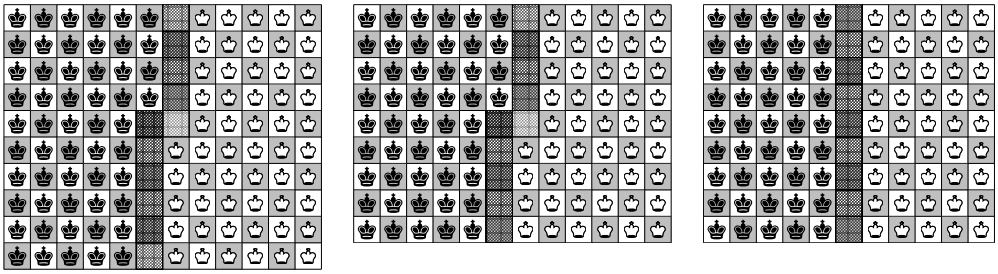

Figura 3.3: Resultados del Corolario 3.1. Caso 1 (izquierda), Caso 2 (centro) y Caso 3 (derecha).

3.3.2. El caso balanceado con caballos en tableros de $m \times n$

En el caso de los caballos recordemos que se tenía una conjetura enunciada en [5]. Escrita de la misma forma en la que se enunció el Corolario 3.1 queda como sigue:

Sea $\phi_{caballo}(n, n)$ la solución óptima que se obtiene al resolver el programa entero (3.50–3.54) para la pieza $caballo$ se tiene que:

Conjetura 3.1

Dado un tablero de ajedrez de $n \times n$. El valor de $\phi_{caballo}(n, n)$ es.

$$\textit{Caso 1: } \phi_{caballo}(n, n) = n\left(\frac{n-2}{2}\right) \quad \textit{si } n \textit{ es par.}$$

$$\textit{Caso 2: } \phi_{caballo}(n, n) = n\left(\frac{n-3}{2}\right) + \frac{n-1}{2} \quad \textit{si } n \textit{ es impar.}$$

Como se mencionó con anterioridad la Conjetura 3.1 no se cumple tanto en el valor como en la forma en la que se deben colocar los caballos en el tablero, además de que no se cumple para valores de $n \leq 6$. A continuación se enunciará el resultado correcto para tableros de ajedrez de $m \times n$ con lo que quedará cubierta la conjetura original.

Teorema 3.1

Dado un tablero de ajedrez de $m \times n$, con $m \leq n$, $m \geq 7$, $n \geq 7$. El valor $\phi_{caballo}(m, n)$ es:

$$\textit{Caso 1: } \phi_{caballo}(m,n) = m\left(\frac{n-2}{2}\right) \textit{ si } n \textit{ es par.}$$

$$\textit{Caso 2: } \phi_{caballo}(m,n) = m\left(\frac{n-3}{2}\right) + \frac{m+1}{2} \textit{ si } m \textit{ y } n \textit{ son impares.}$$

$$\textit{Caso 3: } \phi_{caballo}(m,n) = m\left(\frac{n-3}{2}\right) + \frac{m}{2} \textit{ si } m \textit{ es par y } n \textit{ es impar.}$$

Los valores de $\phi_{caballo}(1,n)$ y $\phi_{caballo}(2,n)$ son $\lfloor \frac{n}{2} \rfloor$ y n respectivamente. Sus demostraciones son triviales. En los casos donde $m = 3,4,5,6$, $\phi_{caballo}(m,n)$ tiene algunos casos patológicos para valores pequeños de n que se resuelven de manera individual. Estos casos pueden ser resueltos mediante la aplicación del Teorema 3.1 con pocas modificaciones. Por ejemplo en el caso de $m = 3$, n debe ser mayor igual que 7 debido a que con $n \leq 6$ existen zonas que algunos caballos no pueden alcanzar.

Proposición 3.1

Dado un tablero de ajedrez de $3 \times n$ *con* $n \geq 7$*, entonces.*

$$\phi_{caballo}(3,n) = 3\left(\frac{n-2}{2}\right) \quad si \quad n \textit{ es par}$$

$$\phi_{caballo}(3,n) = 3\left(\frac{n-3}{2}\right) + 2 \quad si \quad n \textit{ es impar}$$

Demostración. Asumiremos que $b = \phi_{caballo}(3,n)$ y que el número de casillas sin caballo asignado (incoloras) es al menos 6 cuando n es par y al menos 5 cuando n es impar. Lo siguiente es demostrar que el número óptimo de caballos blancos es $\phi(3,n)$. Debido al tamaño de b, existe en el tablero al menos una columna con 2 caballos de color negro. Supongamos que esta es la columna con más caballos negros. Debido a esto, el número de casillas sin caballo debe ser al menos 4 pero ello implica que el número de columnas no vacías es estrictamente mayor $n-2$. Por lo que el número de casillas sin caballo debe ser al menos $4+2(n-2)$, lo cual es claramente mayor que 6. Si llenamos la columna con más caballos negros, se tendrán al menos 6 casillas sin caballo. Si repetimos este procedimiento reduciremos el número de columnas no vacías. Debido a que las columnas no vacías tienen un caballo en cada una de sus filas a excepción de la última. El número de casillas sin caballo es al menos 6 para n par y 5 con n impar. Ahora bien, el número de blancos es.

$$w_{opt} = 3n - b - (\text{número de incoloros}) = 3n - 3\left(\frac{n-2}{2}\right) - 6 = 3\left(\frac{n-2}{2}\right) \text{ En el caso par}$$

Que es el mismo número de casillas sin caballo que en nuestra solución. El caso impar es similar. Para verificar que $\phi_{caballo}(3,n)$ es el más grande de todos los mínimos, es necesario

concluir de manera similar a la del Corolario 3.1.

Para poder demostrar el Teorema 3.1 son necesarios las siguientes definiciones y resultados auxiliares.

3.3.3. Lemas auxiliares para el Teorema 3.1

Definición 3.1

Una casilla incolora es aquella que no puede tener asignados caballos de ningún color.

Definición 3.2

Una **columna llena** *(fila) es aquella que tiene un caballo negro en cada una de sus casillas.*

Definición 3.3

Una **columna vacía** *(fila) es aquella que en todas sus casillas no tiene algún caballo asignado.*

Definición 3.4

Una casilla u es cubierta por un caballo negro en la casilla v, si u no tiene caballo y puede ser alcanzada por el caballo negro en un movimiento.

Lema 3.2

Una **columna llena** *(fila) k cubre completamente las columnas (filas) $k+1$ y $k+2$, para $k \in \{1 \dots n-2\}$.*

Demostración. Es fácil ver que si colocamos a los caballos negros en las casillas negras de la columna k, el número de caballos negros en dichas casillas es $\lfloor \frac{m}{2} \rfloor$ y dichos caballos cubren al menos $\lfloor \frac{m}{2} \rfloor$ casillas diferentes en la columna $k+1$ y al menos el mismo número de casillas en la columna $k+2$ además de que las casillas cubiertas deben ser del color opuesto. Entonces en total se tinenen cubiertas m casillas blancas. Ahora bien si colocamos los caballos restantes en las casillas blancas de la columna k cubriremos al menos m casillas negras entre las columnas $k+1$ y $k+2$. Por lo que la columna llena cubre $2m$ casillas entre la columnas $k+1$ y $k+2$. ■

Definición 3.5

Una **columna casi llena** *(fila) es una columna (fila) con 1 o 2 casillas incoloras.*

Lema 3.3

Una **columna casi llena** *k (fila) cubre al menos a $2m-4$ casillas en las columnas $k+1$ y $k+2$ ($2n-4$ en el caso de las filas).*

Demostración. En este caso utilizaremos el hecho de que cada casilla en las columnas $k+1$ y $k+2$ es cubierta dos veces por los caballos de una columna llena k, con la excepción de

la casillas más alta y más baja en la columna $k+2$ que es cubierta soló una vez. Ahora bien si quitamos dos caballos de la columna k podemos dejar descubiertas hasta 4 casillas en las columnas $k+1$ y $k+2$. ■

Denotaremos como una coloración C a una asignación de caballos negros en el tablero y como $N(C)$ al número de casillas incoloras en una coloración C.

A partir de aquí, las posiciones del tablero seran numeradas en orden ascendente de izquierda a derecha y de abajo hacia arriba como se muestra en la Figura 3.4.

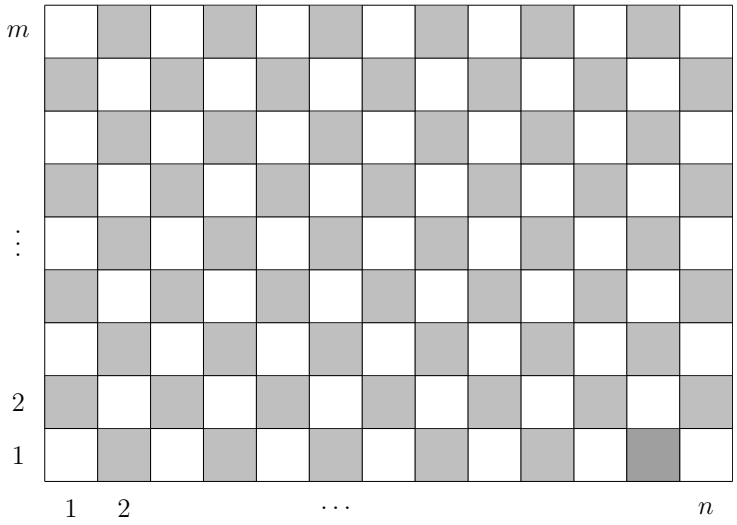

Figura 3.4: Forma de referir posiciones del tablero.

Definición 3.6

Un bloque es un conjunto de columnas (filas) no vacías contiguas.

Definición 3.7

Un bloque vacío es un conjunto de columnas(filas) vacías contiguas.

Lema 3.4

Dada una coloración C con b caballos negros, existe una coloración C', obtenida por la permutación de las columnas (filas) tal que las columnas (filas) llenas y casi llenas están en un sólo bloque y que satisface que $N(C') \leq N(C)$.

Demostración. Sin pérdida de generalidad, supongamos que tenemos una coloración C con todas sus columnas llenas y casi llenas en dos bloques. Las columnas del primer bloque las etiquetamos como $i_1, i_2, \ldots i_p$ y las columnas del segundo bloque como $j_1, j_2, \ldots j_q$ con $i_1 < i_2 \ldots i_p < j_1 < j_2 \ldots j_q$. La parte derecha del bloque izquierdo cubre al menos $2m-4$ casillas en las columnas i_{p+1} e i_{p+2}, lo mismo con la parte izquierda del bloque derecho. Es posible

que el primer bloque y el segundo compartan casillas cubiertas. Si se coloca el segundo bloque enseguida del primero es posible que algunas casillas que podrían albergar caballos blancos entre ambos bloques ahora necesariamente deban ser incoloras. En el peor de los casos se perderán las 4 casillas no cubiertas por el bloque 1 y el bloque 2. Pero han quedado liberadas las columnas j_{q+1} y j_{q+2} completamente por lo que la nueva coloración C' cumple que $N(C') \leq N(C)$ ■.

Lema 3.5

Si una coloración C tiene al menos una columna k casi llena. Tomamos algún caballo que en la coloración C está fuera de la columna k y lo colocamos en dicha columna, la nueva coloración C' satisface que $N(C') \leq N(C)$

Demostración. Esta demostración se divide en los siguientes dos casos:

- Si la columna casi llena tiene un caballo negro en la columna $k+1$ o $k+2$. Debido al Lema 3.3 en las columnas $k+1$ y $k+2$ hay al menos $2(m-2)$ casillas incoloras. Por otro lado, el caballo aislado puede compartir casillas cubiertas con la columna casi llena por su lado izquierdo, pero es claro que dicho caballo cubre al menos 2 casillas ya sea en la columna $k+3$ o $k+4$. Por lo que en el peor de los casos integrar el caballo aislado a la columna casi llena no modifica el número de casillas incoloras.

- Si el caballo aislado está en las columnas $k+3$ o $k+4$, se presenta una situación similar al del caso previo. Si el caballo está en las columnas $k+5, k+6 \ldots n$ el caballo aislado tiene más casillas cubiertas que las que cubriría si se integra a la columna casi llena.

Por lo tanto para ambos casos colocar un caballo aislado en una columna casi llena, provoca que la nueva coloración no aumente en cuanto al número de casillas incoloras. ■

Definición 3.8

Una **columna compacta** *k es una columna no vacía que tiene caballos en las filas $1, 2, \ldots, b_k$.*

Denotaremos con b_i al número de caballos negros en la columna i.

Lema 3.6

Dada una coloración C con $b = \phi_{caballo}(m, n)$ caballos negros, Existe una coloración C' obtenida de compactar todas las columnas no vacías y satisface que $N(C') \leq N(C)$.

Demostración. En este caso sólo se demostrará el caso de n par (impar es análogo). Suponga que la coloración C no tiene columnas llenas o casi llenas, (Si las tuviera el problema es trivial). Por otro lado elijamos la columna con más caballos negros, dicha columna de tener más de $\frac{m}{2}$ caballos negros, de no ser así, debido al tamaño de b se tienen $n-2$ filas no vacías lo que implica al menos una fila casi llena y el problema se vuelve trivial otra vez debido al Lema 3.3. Dado que $m \leq n$ una fila casi llena produce más casillas incoloras que una columna casi llena.

Ahora bien, si seleccionamos la columna k con más caballos negros, esta columna produce en su lado izquierdo al menos $(b_k - b_{k-1}) + (b_k - b_{k-2})$ casillas incoloras y en el lado derecho

al menos $(b_k - b_{k+1}) + (b_k - b_{k+2})$ de estas casillas. Por lo que $N(C) \geq 4b_k - (b_{k-1} + b_{k-2} + b_{k+1} + b_{k+2}) \geq 0$. Si compactamos las columnas como en la Figura 3.5 el número de casillas incoloras no incrementa ya que de ambos lados de la columna k se tiene al menos el mismo número de casillas incoloras. ■

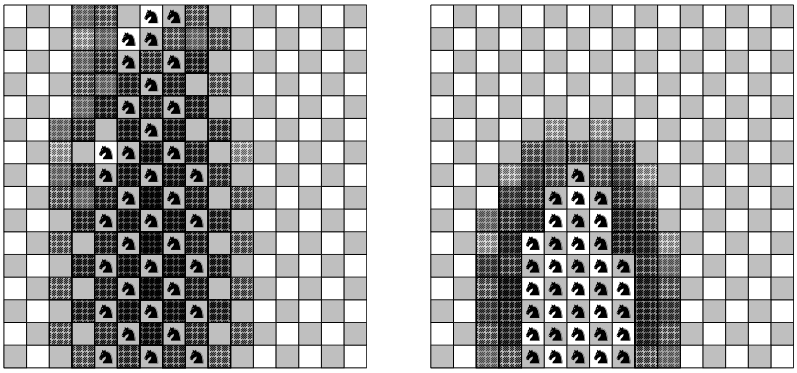

Figura 3.5: Compactación de columnas (Lema 3.6).

Lema 3.7

Dada una coloración C compacta con $b = \phi_{caballo}(m, n)$ caballos negros. Existe una coloración C' tal que todas las columnas no vacías están en orden descendente (respecto al número de caballos negros) y cumple que $N(C') \leq N(C)$.

Demostración. Dada una coloración compacta C. Seleccionamos la columna i con la b_i más grande y la colocamos lo más a la izquierda posible, seguido por la segunda más grande en cuanto a su número de caballos negros y así sucesivamente tal como en la Figura 3.6, tal que mantengan un orden descendente en relación al número de caballos negros. Sabemos por el Lema 3.6 que $N(C) \geq 4b_i - (b_{i-1} + b_{i-2} + b_{i+1} + b_{i+2}) + (b_{i-1} + b_{i-2} + b_{i+1} + b_{i+2}) = 4b_i$, tras el ordenamiento, la nueva coloración C' tiene $N(C') \geq 2b_i + (b_i - b_{i+1}) + (b_i - b_{i+2}) + (b_{i+1} - b_{i+3}) + (b_{i+2} - b_{i+4}) + b_{i+3} + b_{i+4} = 4b_i$ casillas incoloras, por lo que $N(C') \leq N(C)$. ■

Lema 3.8

Dada una coloración compacta y ordenada C con $b = \phi_{caballo}(m, n)$ caballos negros. Existe una coloración C' obtenida de llenar todas las columnas no vacías de tal manera que $b_1 = b_2 = b_3 = \ldots = b_k$, donde la columna 1 es la columna con más caballos negros y k es la última columna que cumple la igualdad, dicha coloración satisface que $N(C') \leq N(C)$.

Demostración. Como la coloración es compacta y ordenada sabemos que tenemos $N(C) \geq (b_1 - b_2) + (b_1 - b_3) + \sum_{i=4}^{k'}(b_i - b_{i-2}) = 2b_1 - b_2 - b_3 + \sum_{i=2}^{k'} b_i - \sum_{i=2}^{k'-2} b_{i+2}$ con k' el número de columnas no vacías

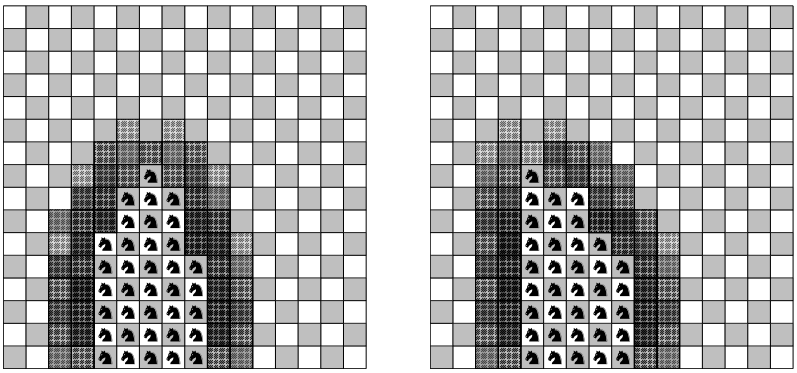

Figura 3.6: Ordenamiento de las columnas (Lema 3.7).

Expandiendo

$$N(C) \geq 2b_1 + (b_2 + b_3 + \ldots + b_{k'}) - (b_4 + b_5 + \ldots + b_{k'-1} + b_{k'}) - b_2 - b_3 + b_{k'-1} + b_{k'} = 2b_1$$

Por el llenado y completado de las columnas tal que $b_1 = b_2 = b_3 = \ldots = b_k$ como se muestra en la Figura 3.7 claramente las diferencias $b_1 - b_2 \geq 0$, $b_1 - b_3 \geq 0 \ldots b_i - b_{i+2} \geq 0$ para toda $i \in \{1, 2, \ldots, k\}$. Es claro que en C algunas de estas diferencias son estrictamente positivas por lo que en el peor de los casos la nueva coloración tiene el mismo número de casillas incoloras $N(C') \geq 2b_1$. ■

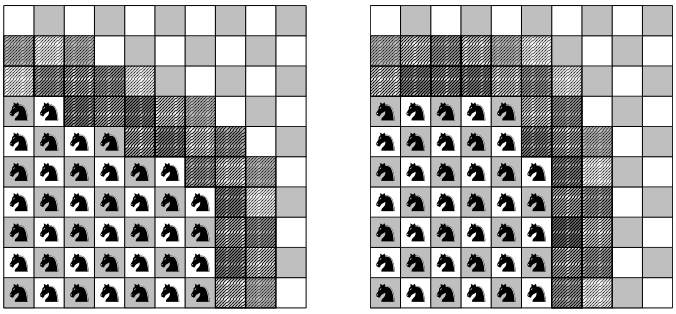

Figura 3.7: Completando columnas (Lema 3.8).

Lema 3.9

Dada una coloración C con b caballos negros. Si C tiene columnas (filas) vacías y obtenemos una coloración C' al colocar todas las columnas (filas) vacías en un sólo bloque de columnas (filas) vacías, la nueva coloración C' satisface $N(C') \leq N(C)$.

Demostración. Sin pérdida de generalidad, supongamos que las columnas vacías están en dos bloques. Sea b_i el número de caballos negros de la última columna no vacía antes del primer

bloque vacío. Es claro que los vértices incoloros estarán en las columnas $i+1$ e $i+2$. Para el segundo bloque de manera similar al Lema 3.5, es posible que las columnas no vacías antes del segundo bloque incoloro compartan vértices cubiertos con la columna i. Por otro lado denotemos como j a la última columna no vacía antes del segundo bloque vacío. Note que dicha columna genera vértices incoloros en las columnas $j+1$ y $j+2$. Es fácil ver que si se juntan las secciones no vacías puede suceder que algunas casillas que eran potencialmente blancas necesariamente se vuelvan incoloras, dado que al menos había una columna vacía en el primer bloque vacío, se tiene en la nueva coloración C' al menos una columna vacía no cubierta lo cual genera más casillas que pueden tener caballos blancos y por lo tanto $N(C') \leq N(C)$. ■

Lema 3.10

Dada una coloración C con $b = \phi_{caballo}(m,n)$ caballos negros. Existe una coloración C' con al menos una columna llena tal que $N(C') \leq N(C)$.

Demostración. Dada una coloración C cualquiera, con la aplicación de los Lemas 3.4, 3.5, 3.6, 3.7, 3.8 y 3.9 podemos convertir esa coloración C en una formada por un bloque con $N(C) \geq 2b_{c_1} + 2b_{r_1}$, donde b_{c_1} es la columna con más caballos negros y b_{r_1} la fila con más caballos negros. Claramente $N(C) \geq 2(b_{c_1} + b_{r_1}) \geq 2m$. Si llenamos la columna más grande hasta tener una columna llena, el número de casillas incoloras no aumenta. Posteriormente volvemos aplicar el Lema 3.8 y obtendremos un bloque donde hay al menos una columna c_1 con m caballos negros y al menos una fila r_1' con $b_{r_1'}$ caballos negros. Está claro que $b_{r_1'} \leq b_{r_1}$ y las casillas cubiertas por la fila r_1 ya están cubiertas por la última columna llena, por lo que se mantiene $N(C') \geq 2m$ por lo tanto $N(C') \leq N(C)$. ■

Lema 3.11

Dada una coloración C con al menos una columna llena, entonces $N(C) \geq 2m-1$ o $N(C) \geq 2m$ dependiendo de la paridad.

Demostración. En este caso existen 2 posibilidades como se puede ver en la Figura 3.8. Sea k la última columna no vacía en el tablero y sea b_k el número de caballos negros en dicha columna.

- Caso 1: Las columnas $k+1$ y $k+2$ tienen al menos b_k casillas incoloras.
- Caso 2: Las columnas $k+1$ y $k+2$ tienen al menos $m - b_k$ casillas incoloras.

Claramente el segundo caso sólo es posible cuando m y n son impares. Debido al Lema 3.9 es posible juntar la columna k con la última columna llena, permitiendo que esta nueva coloración C' cumpla $N(C') \leq N(C)$. Debido al efecto de la columna llena. En la columna k tenemos $m - b_k$ casillas incoloras y m casillas incoloras en la columna $k+1$. Por el efecto del la última columna no vacía tenemos: o bien, al menos b_k o $m - b_k$ casillas incoloras en la columna $k+2$ dependiendo de la paridad del tablero. En total tenemos

- Caso 1: $N(C) \geq m - b_k + m + b_k = 2m$.

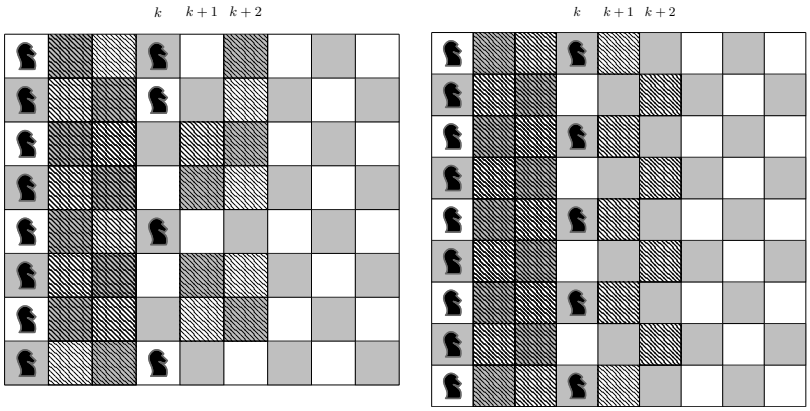

Figura 3.8: Los dos posibles casos dependiendo de la paridad del tablero.

- Caso 2: $N(C) \geq m - b_k + m + m - b_k = 3m - 2b_k$, dado que $b_k \geq \frac{m+1}{2}$ entonces $N(C) \geq 2m - 1$

■

Lema 3.12

Dada una coloración C con $b = \phi_{caballo}(m, n)$ caballos negros entonces $N(C) \geq 2m$ or $N(C) \geq 2m - 1$.

Demostración. Debido al tamaño de b por el Lema 3.10 es posible construir una solución C' equivalente que contiene al menos una columna llena. Por el efecto del Lema 3.11 esa solución tiene $N(C) \geq 2m$ o $N(C) \geq 2m - 1$ dependiendo de la paridad ■

Conjunto incoloro mínimo

A continuación se propone un conjunto de soluciones que cumplen exactamente la cota inferior en cuanto al número de incoloros 3.1. Las configuraciones pueden verse en la Figura 3.9.

Caso	m	n	casillas incoloras
1	par	par	$2m$
2	impar	par	$2m$
3	par	impar	$2m$
4	impar	impar	$2m - 1$

Tabla 3.1: La solución propuesta

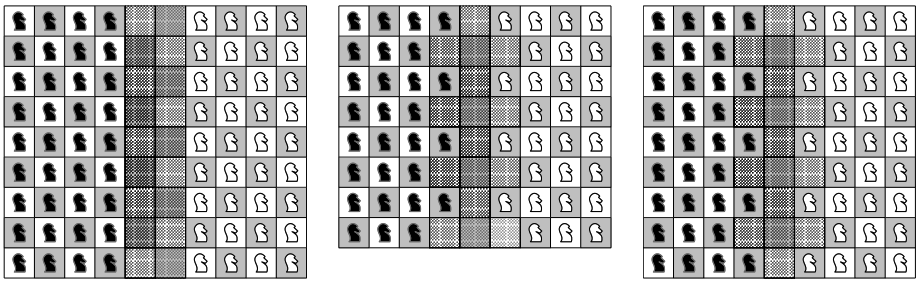

Figura 3.9: Ejemplos de las soluciones propuestas, Caso 1,2 (izquierda), Caso 3 (centro), Caso 4 (derecha).

3.3.4. Conclusión del Teorema 3.1

Ahora podemos escribir una fórmula para calcular el número óptimo de blancos dados $b = \phi_{caballo}(m, n)$ caballos negros.

$$w_{opt} = mn - (b = \phi_{caballo}(m, n)) - \begin{cases} 2m & \text{Casos 1,2 y 3} \\ 2m - 1 & \text{Caso 4} \end{cases} \tag{3.56}$$

Para el Caso 1 (m par, n par), calculamos el número óptimo de caballos blancos que pueden ser colocados en el tablero. Como nuestra solución es una cota justa sobre el número de incoloros, entonces es óptima en cuanto al número de caballos blancos. Tenemos que

$$w_{opt} = mn - m\left(\frac{n-2}{2}\right) - 2m = m\left(\frac{n-2}{2}\right)$$

Por la definición de BWC-Balanceado tenemos que $\text{mín}(b, w_{opt}) = b$ porque $b = w_{opt}$. Adicionalmente se necesita el más grande de todos los mínimos. Si incrementamos el valor de b en k unidades y $\phi(m, n)$ no es óptima, el valor de $\text{mín}(b, w_{opt})$ incrementará. Sea w^k_{opt} el número óptimo de caballos blancos dados $b + k$ caballos negros, con $k \geq 1$.

$$w^k_{opt} = mn - m\left(\frac{n-2}{2}\right) - k - 2m = m\left(\frac{n-2}{2}\right) - k$$

ahora

$$\text{mín}(b + k, w^k_{opt}) = w^k_{opt}$$

y por tanto

$$\text{máx}(b, w^k_{opt}) = b$$

■

La conclusión en los casos 2, 3 y 4 es análoga.

3.3.5. El caso balanceado con reinas en tableros de $m \times n$

Recordemos que el problema original de Berge estaba planteado con las reinas. En este caso no se cuenta con una demostración sobre la la optimalidad de la forma de acomodo. Con lo que sí se cuenta es con un factor de ocupación del tablero. Es decir, se tiene un $\kappa \in [0, 1]$ tal que $\phi_{reina}(m, n) = \kappa mn$. En el caso particular de $m = n$ Berend *et al.*, obtuvieron de manera numérica un valor de $\kappa = 0.132$, es decir $\phi_{reina}(n, n) = 0.132n^2$. Dicho valor se alcanza mediante la forma de acomodo de la Figura 3.10. Ellos mencionan que es un comportamiento asintótico para tableros con un valor muy grande de n aunque no mencionan qué tan grande.

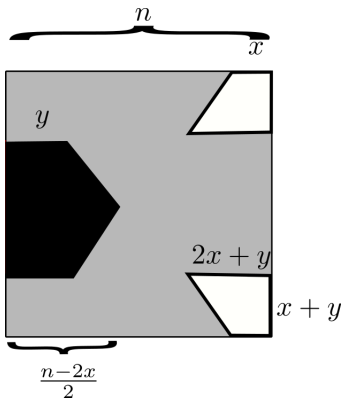

Figura 3.10: Forma de acomodar las reinas en un tablero cuadrado .

Trataremos de corroborar que el óptimo tiene una forma al menos parecida a la de la Figura 3.10 o en su defecto mostraremos la forma que conduce al óptimo. También se tratará de ver de manera experimental que el valor de $\kappa \approx 0.132$. Para ello partiremos del programa lineal entero (3.50–3.54) para la pieza reina. La Tabla 3.2 muestra los resultados obtenidos para algunos tamaños de tableros.

Como se puede observar en la Tabla 3.2 el valor obtenido de manera experimental del valor de κ es cercano al valor propuesto en [5]. En este sentido hay que puntualizar dos detalles sobre el valor de κ. La primera es que el valor de κ se conjetura como cota inferior y en segundo se cree que el óptimo está más cercano de ese valor que de una cota superior de 0.25 ya que 0.25 es el valor de κ para el caso de las torres.

Por otro lado cabe destacar que los valores obtenidos de manera experimental de κ fue mediante la solución del modelo durante $1hr$ en Gurobi. por lo que la solución encontrada no necesariamente es la óptima pero sí es cercana al valor de κ predicho.

Se puede revisar la forma de acomodo que tienen las soluciones. Si el óptimo fuera único implicaría que la forma que se debería de observar es la conjeturada por Berend *et al.* Sin embargo, hay múltiples soluciones incluyendo algunas que no coinciden con la forma de señalada en la Figura 3.10.

En la Figura 3.11 se pùede observar algunas soluciones para tamaños de n, es posible distinguir la forma de la conjetura de Berend *et al.* Como se acaba de mencionar, las soluciones

n	$\phi_{reina}(n,n)$	κ
7	7	0.142
8	9	0.140
9	12	0.148
10	14	0.140
11	17	0.140
12	21	0.145
13	24	0.142
14	27	0.137
15	30	0.133
16	34	0.132
17	40	0.138
18	43	0.132

n	$\phi_{reina}(n,n)$	κ
19	47	0.130
20	55	0.137
21	59	0.131
22	63	0.130
23	73	0.137
24	79	0.137
25	87	0.139
26	95	0.140
27	91	0.124
28	102	0.130
29	102	0.121
30	111	0.123

Tabla 3.2: Resultados experimentales para el caso de las reinas balanceadas en tableros de $n \times n$.

no son únicas por lo que no necesariamente ésta es la única opción de asignación. En la Figura 3.12 se pueden observar dos soluciones con el mismo valor de κ pero con una forma diferente.

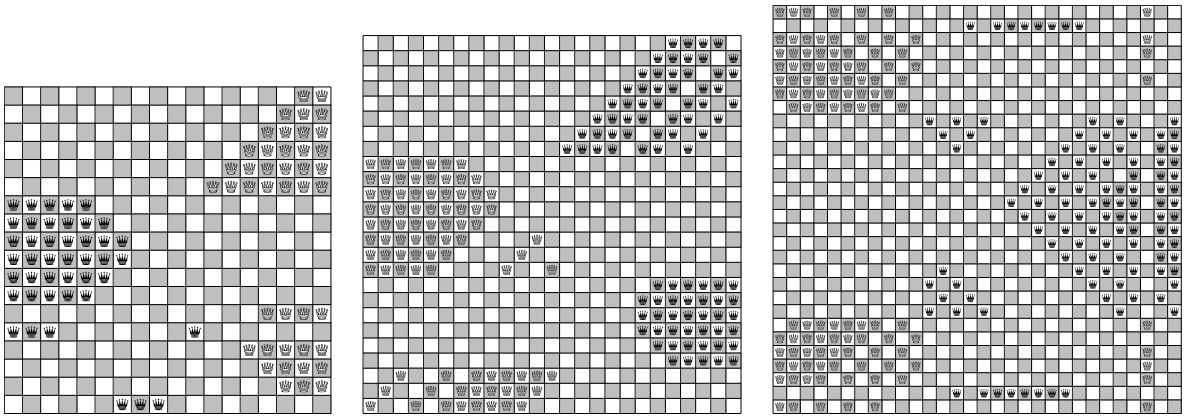

Figura 3.11: Resultados experimentales para $n = 18$, $n = 25$ y $n = 30$.

Por simple observación se puede conjeturar que es posible llevar las coloraciones parecidas a la forma propuesta a una coloración todavía más parecida sin provocar que el valor de κ disminuya, de manera similar a como se hizo con los caballos en la subsección anterior.

En cuanto a los tableros rectangulares se tienen los resultados del valor de κ resumidos en la Tabla 3.3.

Con respecto a la forma que tienen las asignaciones de reinas blancas y negras en tableros rectangulares. Podemos observar que entre más pequeña es la relación $\frac{m}{n}$ la asignación es más

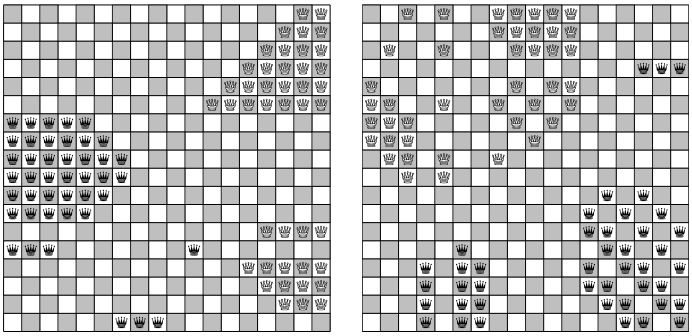

Figura 3.12: Resultados experimentales con diferente forma para $n = 18$.

m/n	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
7	0.143	0.143	0.159	0.157	0.169	0.179	0.176	0.184	0.190	0.196	0.193	0.198	0.203	0.200	0.204	0.208	0.205	0.208	0.211	0.209	0.212	0.214	0.217	0.214
8		0.141	0.139	0.150	0.159	0.167	0.173	0.179	0.183	0.188	0.191	0.194	0.197	0.200	0.202	0.205	0.207	0.208	0.205	0.212	0.213	0.214	0.216	0.217
9			0.148	0.156	0.152	0.157	0.162	0.167	0.178	0.174	0.183	0.185	0.187	0.189	0.196	0.197	0.198	0.199	0.204	0.205	0.206	0.206	0.207	0.211
10				0.140	0.145	0.150	0.154	0.157	0.167	0.169	0.171	0.178	0.184	0.185	0.190	0.191	0.191	0.196	0.196	0.200	0.200	0.204	0.207	0.207
11					0.140	0.136	0.147	0.156	0.158	0.159	0.166	0.172	0.177	0.177	0.186	0.186	0.190	0.189	0.193	0.196	0.199	0.198	0.201	0.203
12						0.146	0.135	0.149	0.150	0.156	0.162	0.167	0.167	0.175	0.171	0.178	0.181	0.188	0.190	0.192	0.194	0.193	0.195	0.197
13							0.142	0.137	0.144	0.149	0.154	0.158	0.162	0.169	0.172	0.175	0.177	0.183	0.185	0.186	0.188	0.190	0.191	0.195
14								0.138	0.143	0.143	0.143	0.147	0.158	0.161	0.163	0.169	0.174	0.173	0.177	0.181	0.185	0.186	0.187	0.190
15									0.133	0.138	0.133	0.144	0.151	0.153	0.159	0.161	0.168	0.169	0.173	0.174	0.180	0.181	0.184	0.187
16										0.133	0.140	0.146	0.138	0.147	0.152	0.156	0.163	0.167	0.168	0.168	0.174	0.176	0.179	0.183
17											0.138	0.137	0.139	0.141	0.146	0.150	0.153	0.162	0.162	0.165	0.170	0.170	0.176	0.178
18												0.133	0.135	0.139	0.146	0.146	0.150	0.153	0.156	0.162	0.167	0.169	0.170	0.174
19													0.130	0.134	0.138	0.146	0.144	0.149	0.149	0.158	0.160	0.165	0.165	0.170
20														0.138	0.138	0.141	0.139	0.148	0.146	0.156	0.157	0.161	0.162	0.167
21															0.134	0.143	0.139	0.143	0.145	0.150	0.155	0.158	0.158	0.159
22																0.130	0.136	0.140	0.138	0.143	0.148	0.148	0.154	0.156
23																	0.138	0.138	0.137	0.142	0.134	0.143	0.151	0.151
24																		0.137	0.140	0.138	0.137	0.146	0.142	0.149
25																			0.139	0.123	0.135	0.140	0.131	0.144
26																				0.141	0.142	0.133	0.131	0.145
27																					0.126	0.126	0.119	0.135
28																						0.130	0.113	0.133
29																							0.121	0.128
30																								0.123

Tabla 3.3: Valores experimentales del valor de κ para tableros de reinas de $m \times n$.

parecida a la propuesta por Berend *et al.*. En la Figura 3.13 se pueden ver algunos ejemplos de asignaciones.

De la Tabla 3.3 se puede interpretar que entre mas se acerca la relación $\frac{m}{n}$ a 1, el valor de κ es más cercano a 0.132, mientras que si dicha relación se cerca mas a 0, el valor de κ es más cercano a 0.25. Tome como ejemplo los tableros de dimensiones 7×30 y 11×30.

3.4. PAC con $k > 2$ colores

En el caso del PAC cuando $k = 3$ lo denominaremos RGBC en el caso del problema de decisión y ORGBC en el caso del problema de optimización, haciendo alusión al uso de los colores rojo verde y azul preservando las siglas en inglés para mantener la idea del BWC. En este caso mostraremos el modelo de programación entera para 3 colores con 2 fijos y maximizando un tercero.

Para obtener un programa entero para el ORGBC, partiremos de un programa similar al

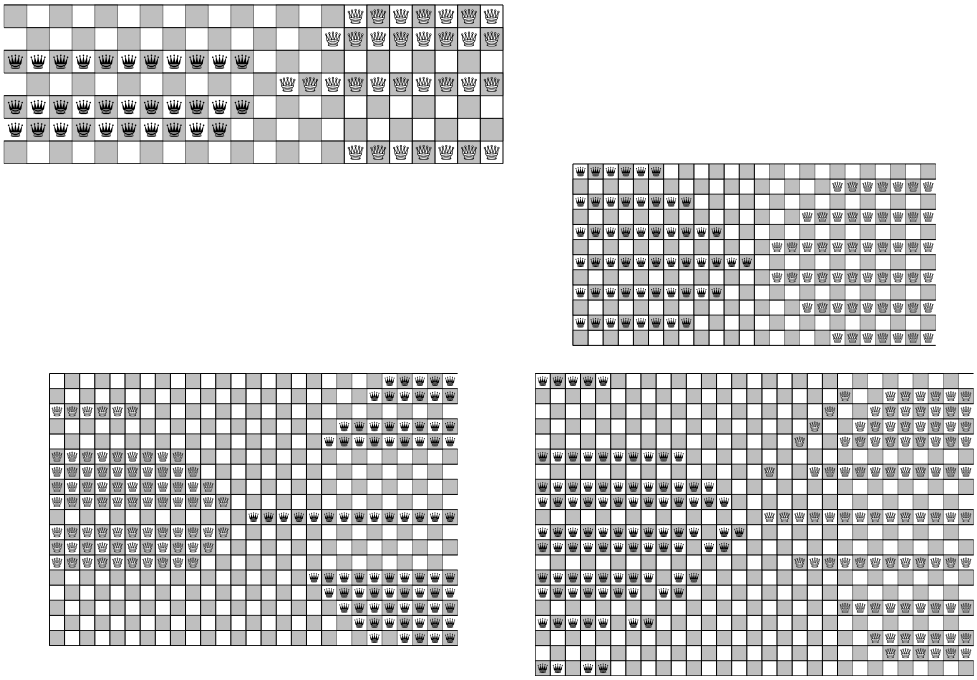

Figura 3.13: Ejemplos de algunos tableros rectangulares de reinas.

programa lineal entero (2.8–2.12), extendiéndolo a un color adicional. Posteriormente realizaremos un procedimiento similar al del programa lineal entero (3.1–3.4) para reducir el número de restricciones.

$$x_v = \begin{cases} 1 & \text{si al vértice } v \text{ se le asigna el color } \textbf{rojo} \\ 0 & \text{en otro caso.} \end{cases}$$

$$y_v = \begin{cases} 1 & \text{si al vértice } v \text{ se le asigna el color } \textbf{verde} \\ 0 & \text{en otro caso.} \end{cases}$$

$$z_v = \begin{cases} 1 & \text{si al vértice } v \text{ se le asigna el color } \textbf{azul} \\ 0 & \text{en otro caso.} \end{cases}$$

$$
\begin{aligned}
z &= \text{máx} \sum_{v \in V} z_v && (3.57)\\
\sum_{v \in V} x_v &= r, && (3.58)\\
\sum_{v \in V} y_v &= g, && (3.59)\\
x_v + y_v &\leq 1 \ \forall \ v \in V && (3.60)\\
x_u + y_v &\leq 1 \ \forall \ \{u, v\} \in E && (3.61)\\
x_v + z_v &\leq 1 \ \forall \ v \in V && (3.62)\\
x_u + z_v &\leq 1 \ \forall \ \{u, v\} \in E && (3.63)\\
y_v + z_v &\leq 1 \ \forall \ v \in V && (3.64)\\
y_u + z_v &\leq 1 \ \forall \ \{u, v\} \in E && (3.65)\\
x_v, y_v &\in \{0, 1\} \ \forall \ v \in V && (3.66)
\end{aligned}
$$

Aplicando la suma de todas las restricciones de un vértice dado para un color específico nos queda el siguiente programa lineal entero.

$$
\begin{aligned}
z &= \text{máx} \sum_{v \in V} z_v && (3.67)\\
\sum_{v \in V} x_v &= r && (3.68)\\
\sum_{v \in V} y_v &= g && (3.69)\\
\sum_{s \in N[v]} y_s &\leq |N[v]|(1 - x_v) \quad \text{para toda } v \in V && (3.70)\\
\sum_{s \in N[v]} z_s &\leq |N[v]|(1 - x_v) \quad \text{para toda } v \in V && (3.71)\\
\sum_{s \in N[v]} z_s &\leq |N[v]|(1 - y_v) \quad \text{para toda } v \in V && (3.72)\\
x_v, y_v, z_v &\in \{0, 1\}. && (3.73)
\end{aligned}
$$

La función objetivo 3.67 maximiza el número de vértices azules mientras que las restricciones 3.68 y 3.69 fijan el número de vértices con color rojo y verde respectivamente. Los conjuntos de restricciones 3.70 al 3.71 representan todas las posibles formas en las que un vértice y su vecindario pudiese quedar bicoloreado, es decir si un vértice v es rojo su vecindario cerrado no podría ser ni verde ni azul o en el caso de que el vértice v sea verde su vecindario cerrado no podría ser azul ni rojo.

Para el ejemplo de la Figura 3.1 tenemos el siguiente modelo de programación lineal entera

Ejemplo 3.2

$$
\begin{aligned}
z &= \text{máx } z_1 + z_2 + z_3 + z_4 && (3.74)\\
x_1 + x_2 + x_3 + x_4 &= r && (3.75)\\
y_1 + y_2 + y_3 + y_4 &= g && (3.76)\\
x_1 + y_1 &\leq 1 && (3.77)\\
x_1 + y_2 &\leq 1 && (3.78)\\
x_1 + y_3 &\leq 1 && (3.79)\\
x_1 + y_4 &\leq 1 && (3.80)\\
x_1 + z_1 &\leq 1 && (3.81)\\
x_1 + z_2 &\leq 1 && (3.82)\\
x_1 + z_3 &\leq 1 && (3.83)\\
x_1 + z_4 &\leq 1 && (3.84)\\
y_1 + z_1 &\leq 1 && (3.85)\\
y_1 + z_2 &\leq 1 && (3.86)\\
y_1 + z_3 &\leq 1 && (3.87)\\
y_1 + z_4 &\leq 1 && (3.88)\\
&\vdots && (3.89)\\
x_i, y_i, z_i &\in \{0, 1\}, i = 1, 2, 3, 4. && (3.90)
\end{aligned}
$$

Tan sólo para el vértice 1 podemos sumar por separado las desigualdades (3.77–3.80), (3.81–3.84) y (3.85–3.88). Esto nos genera las siguientes tres expresiones:

$$4x_1 + y_1 + y_2 + y_3 + y_4 \leq 4$$

$$4x_1 + z_1 + z_2 + z_3 + z_4 \leq 4$$

$$4y_1 + z_1 + z_2 + z_3 + z_4 \leq 4$$

Después de despejar y factorizar

$$y_1 + y_2 + y_3 + y_4 \leq 4(1 - x_1)$$

$$z_1 + z_2 + z_3 + z_4 \leq 4(1 - x_1)$$

$$z_1 + z_2 + z_3 + z_4 \leq 4(1 - y_1)$$

que son expresiones de la forma de las desigualdades 3.70–3.72.

Algo que se puede destacar de estos modelos es que el programa lineal entero (3.67–3.73) crecerá en número de restricciones a razón de $O\left(\binom{k}{2}\right)$ de donde k es el número de clases de color que se desean en el problema.

3.4.1. Modelos para el PAC con $k > 2$ colores

Una versión general del PAC en su versión de optimización para k colores sería la siguiente:

$$x_{v,i} = \begin{cases} 1 & \text{si al vértice } v \text{ se le asigna el color } i \\ 0 & \text{en otro caso.} \end{cases}$$

$$z = \text{máx} \sum_{v \in V} x_{v,k} \tag{3.91}$$

$$\sum_{v \in V} x_{v,i} = c_i, \text{ para toda } i = 1, 2, \ldots, k-1 \tag{3.92}$$

$$\sum_{s \in N[v]} x_{s,i} \leq |N[v]|(1 - x_{v,j}) \quad \text{para toda } v \in V,\ i, j \in \{1, \ldots, k\} \text{ e } i \neq j \tag{3.93}$$

$$x_{v,i} \in \{0, 1\}, \text{ para toda } v \in V \text{ e } i \in \{1, \ldots, k\}. \tag{3.94}$$

En este caso la función objetivo 3.91 maximiza el número de vértices con color k con los $k-1$ colores anteriores con un número fijo de vértices como lo muestra el conjunto de restricciones 3.92. El conjunto de restricciones 3.93 toma todas las combinaciones de dos colores de un conjunto de k de ellos y no permite que dado un vértice v ni el vértice ni su vecindario cerrado sean bicoloreados para cualquier pareja posible de colores.

Con respecto al modelo de programación no lineal entero de factibilidad, se puede extender a más de dos colores. El modelo no lineal entero para el PAC con $k > 2$ es el siguiente.

$$z = \text{mín} \sum_{i,j \in \{1 \ldots k | i \neq j\}} \sum_{\{u,v\} \in E} (x_{u,i} x_{v,j} + x_{v,i} x_{u,j}) + \sum_{i,j \in \{1 \ldots k | i \neq j\}} \sum_{v \in V} x_{v,i} x_{v,j} \tag{3.95}$$

$$\sum_{v \in V} x_{v,i} = c_i \text{ para toda } i \in \{1 \ldots k\} \tag{3.96}$$

$$x_{v,i} \in \{0, 1\} \text{ para toda } v \in V \text{ e } i \in \{1, \ldots, k\}. \tag{3.97}$$

Este último modelo también se puede utilizar para resolver el PPG si agregamos el supuesto de que.

$$\sum_{i \in \{1 \ldots k\}} c_i = |V|$$

De manera similar si utilizamos un balance generalizado, las c_i pasan a ser variables y la nueva función objetivo es.

$$\text{máx}\{\text{mín}\{c_i | i \in \{1, \ldots, k\}\}\}$$

3.5. Método por etapas

Basándonos en el modelo de programación entera (3.91–3.94). Pensemos en el caso de 3 colores. Una vez colocado el color 1, el espacio libre para el color 2 y 3 son los vértices que no son adyacentes a los vértices con color 1 asignado. Por esta razón una posible estrategia es retirar de la gráfica a los vértices con color 1 y a sus vecinos (incoloros). Posteriormente aplicar algún método (exacto o heurístico) para los dos colores restantes.

Un ejemplo se puede ver en la Figura 3.14 donde primero se asigna el color rojo como si el problema fuera con sólo dos colores (rojo y verde), posteriormente se retiran de la gráfica los vértices rojos y sus vecinos (incoloros) con sus respectivas aristas. En la gráfica restante se repite el procedimiento para dos colores.

La solución obtenida al resolver el mismo ejemplo a través del programa lineal entero (3.67–3.72) se puede ver en la Figura 3.15.

Como se puede observar es exactamente la misma solución que generó el método por etapas. Sin embargo esto no es garantía de que el método funcione siempre con 3 colores o más. Es una manera de abordar el problema cuando se tienen más de dos colores y hasta el momento no se conocía alguna técnica para este problema en gráficas generales. La Tablas 3.4 y 3.5 muestran dos de las instancias propuestas en [6] que originalmente eran para dos colores ahora adaptadas para tres, se muestran los dos colores fijos y el número obtenido de vértices del tercer color. En dichas tablas se puede observar cierta discrepancia entre el método directo y el método por etapas. En la tabla 3.4 el método directo muestra la solución óptima, mientras que el método por etapas muestra el resultado generado al obtener el óptimo en cada etapa (obtenida a partir de los resultados de Berend *et al.*, para tableros de ajedrez toroidales con reyes).

La discrepancia no resulta ser tan grande y mejor aún los tiempos de ejecución son sumamente más bajos en el método por etapas. Mientras que en el método directo se requirió alrededor de 1 hora por instancia (últimas 4 instancias de la primer gráfica en hardware especializado para ser resuelto). En el caso de la gráfica aleatoria el método directo se ejecutó por 1 hora mientras que al método por etapas media hora en cada etapa. En este caso en el método directo se da la solución obtenida después de 1 hr de ejecución en Gurobi mientras que en el método por etapas se da la solución obtenida después de 30 minutos en la primera etapa y 30 minutos en la segunda. La discrepancia en esta ocasión se inclina al método por etapas, esto puede deberse a que es posible que exista más de una solución óptima. Una posible línea de investigación es averiguar qué pasa con el método por etapas y si existen casos en donde la solución sea exacta siempre.

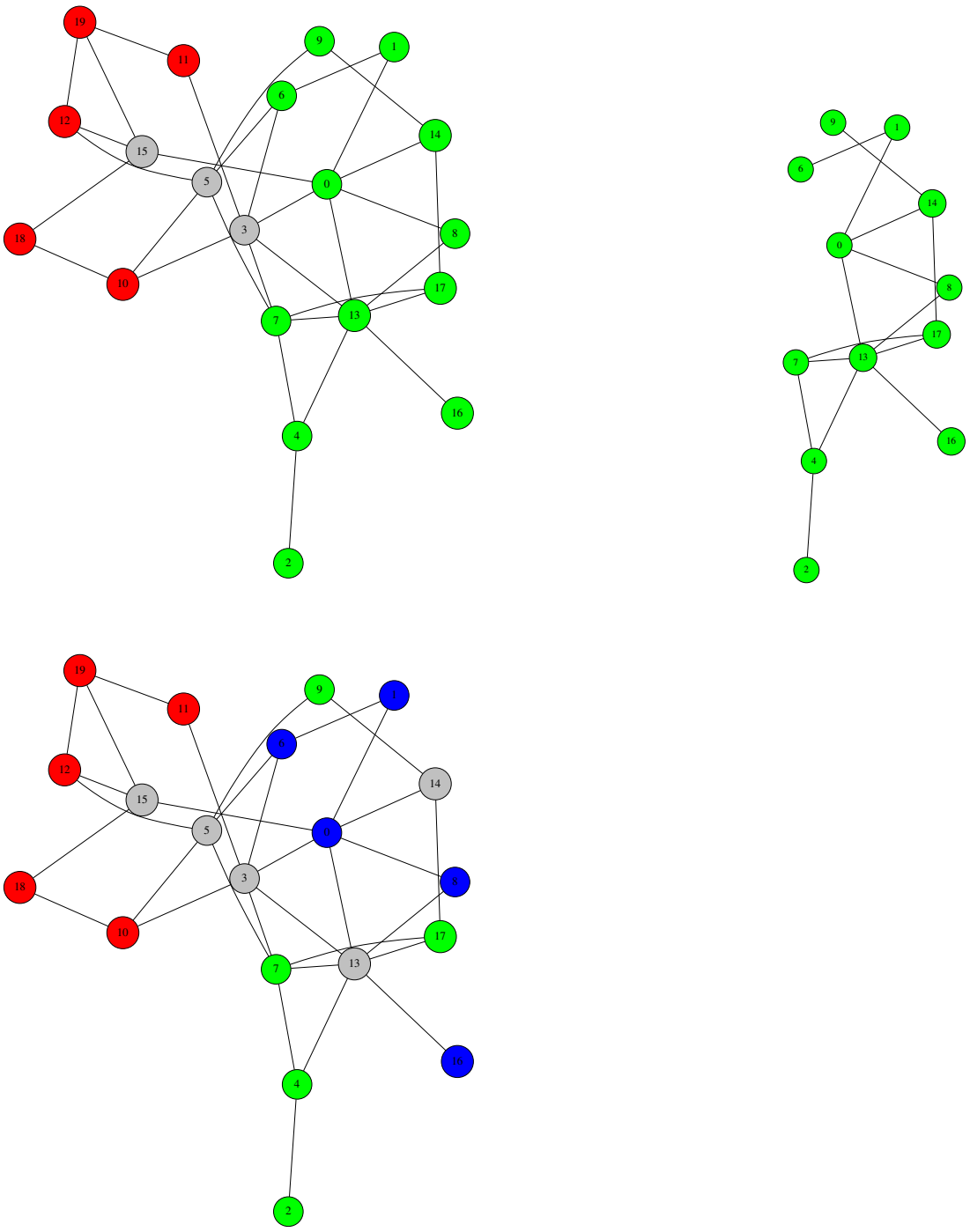

Figura 3.14: Ejemplo del método por etapas.

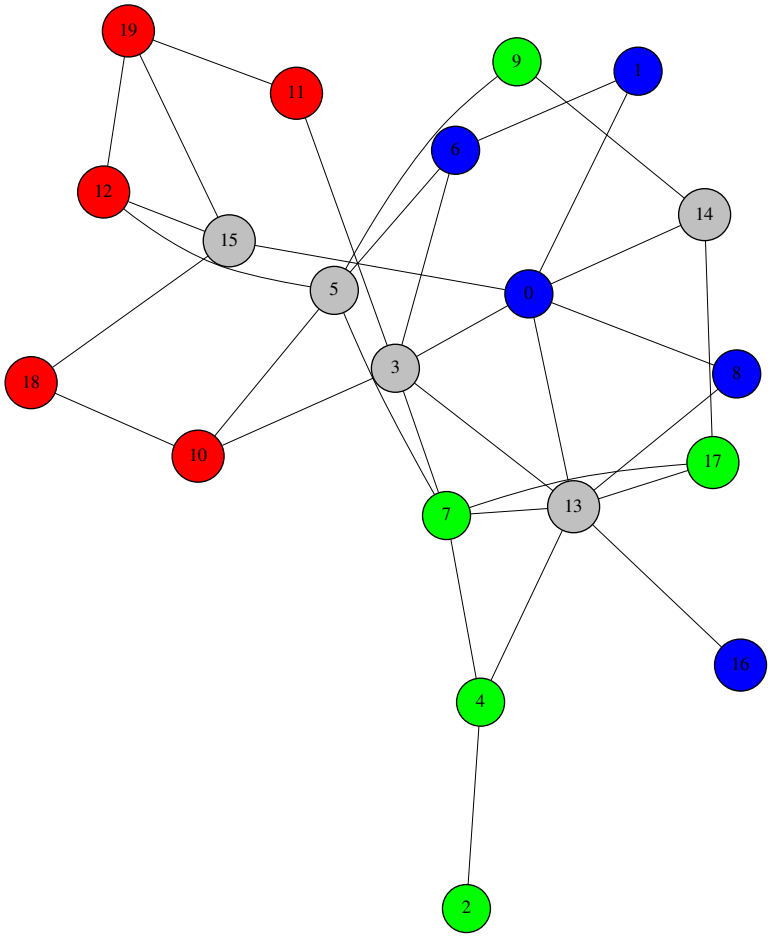

Figura 3.15: Solución obtenida a partir del programa lineal entero (3.67–3.72).

Gráfica	r	g	Directo	Etapas
$C_{15} \boxtimes C_{11}$	10	10	117	117
$C_{15} \boxtimes C_{11}$	10	20	104	103
$C_{15} \boxtimes C_{11}$	10	30	92	92
$C_{15} \boxtimes C_{11}$	10	50	71	71
$C_{15} \boxtimes C_{11}$	10	70	52	52
$C_{15} \boxtimes C_{11}$	20	20	93	93
$C_{15} \boxtimes C_{11}$	20	30	81	78
$C_{15} \boxtimes C_{11}$	20	50	58	57
$C_{15} \boxtimes C_{11}$	20	70	38	38
$C_{15} \boxtimes C_{11}$	30	30	68	68
$C_{15} \boxtimes C_{11}$	30	50	48	48
$C_{15} \boxtimes C_{11}$	30	70	28	28
$C_{15} \boxtimes C_{11}$	50	50	28	28
$C_{15} \boxtimes C_{11}$	50	70	11	11
$C_{15} \boxtimes C_{11}$	70	70	0	0

Tabla 3.4: Ejemplo comparativo entre el método por etapas y aplicando el programa lineal entero para tres colores.

Gráfica	r	g	Directo	Etapas
$rnd_{500,0.1}$	10	10	122	132
$rnd_{500,0.1}$	10	20	69	78
$rnd_{500,0.1}$	10	30	47	46
$rnd_{500,0.1}$	10	50	25	31
$rnd_{500,0.1}$	10	70	17	22
$rnd_{500,0.1}$	10	100	11	14
$rnd_{500,0.1}$	20	20	44	46
$rnd_{500,0.1}$	20	30	33	32
$rnd_{500,0.1}$	20	50	9	18
$rnd_{500,0.1}$	20	70	5	11
$rnd_{500,0.1}$	20	100	-	5
$rnd_{500,0.1}$	30	30	11	19
$rnd_{500,0.1}$	30	50	2	9
$rnd_{500,0.1}$	30	70	-	3
$rnd_{500,0.1}$	30	100	-	-
$rnd_{500,0.1}$	50	50	-	-
$rnd_{500,0.1}$	50	70	-	-
$rnd_{500,0.1}$	50	100	-	-
$rnd_{500,0.1}$	70	70	-	0
$rnd_{500,0.1}$	70	100	-	0
$rnd_{500,0.1}$	100	100	-	0

Tabla 3.5: Ejemplo comparativo entre el método por etapas y aplicando el programa lineal entero para tres colores.

Capítulo 4
Anticoloraciones en gráficas y heurísticas

Como ya se ha mencionado en capítulos previos el PAC es NP-Completo en gráficas generales aún cuando se trata del caso de dos colores. Dado que tampoco se conocen algoritmos de aproximación o siquiera que el problema es aproximable, lo hace un buen candidato al uso de técnicas heurísticas y en particular justifican el uso de meta-heurísticas. En las siguientes secciones se describirán dos técnicas meta-heurísticas. La primer técnica a utilizar es la conocida como GRASP que mostró resultados de buena calidad comparados con los de Búsqueda Tabú [6]. Una parte importante de la ejecución del GRASP, la calibración de parámetros fue hecha al tanteo por lo que se recrea el algoritmo y se calibra automáticamente por medio de irace. Por otro lado también se presenta un Algoritmo Genético en dos variantes distintas (mutación constante y mutación variable) y se muestra que es competitivo en comparación con la Búsqueda Tabú. Posteriormente se aborda el caso del PAC para más de dos colores y lo necesario para implementar meta-heurísticas para este caso. Por último se muestra el posible enfoque multiobjetivo de este problema y las posibilidades a futuro con el problema. Las contribuciones en este ámbito se pueden resumir en.

- GRASP + calibración automática para OBWC.
- Algoritmo genético para OBWC.
- Calibración automática del Algoritmo genético.
- Función de aptitud y corrección para más de 3 colores.
- Propuesta multiobjetivo.

4.1. GRASP para OBWC con calibración automática

Como ya se mencionó el GRASP consta de dos etapas. La primera relacionada con una construcción de la solución inicial de manera glotona aleatorizada o semi-glotona variando qué tan glotona o qué tan aleatoria es la solución por medio de un parámetro $\alpha \in [0, 1]$. Creando una lista de posibles candidatos a ser agregados a la solución. En el caso particular del OBWC la construcción se da mediante el Algoritmo 4.1.

La codificación de una solución se da a partir de un vector binario x donde $x[v] = 1$ representa si el vértice v tiene asignado el color negro. En este caso los vértices incoloros y blancos se colocan usando el Algoritmo 4.2 y se cuenta cuantos blancos se pueden colocar el vector de vértices blancos es un vector temporal que sólo está presente en la evaluación.

Algoritmo 4.1 Construcción de solución inicial para el GRASP

Entrada: $G = (V, E)$, b, α
Salida: s_0 Solución inicial factible del OBWC

1: $s_0 = \emptyset$
2: **para** $k = 1$ **hasta** b **hacer**
3: $S_{candidatos} = \emptyset$
4: $S_{mejores} = \emptyset$
5: **para cada** $v \in V \backslash s_0$ **hacer**
6: $s_0 = s_0 \cup v$
7: $aux = s_0$
8: $evaluar(aux)$
9: $S_{candidatos} \cup aux$
10: $s_0 = s_0 \backslash v$
11: **fin para**
12: $c_{min} = \text{mín}\{S_{candidatos}\}$, $c_{max} = \text{máx}\{S_{candidatos}\}$
13: **para cada** $s_c \in S_{candidatos}$ **hacer**
14: **si** $c_{min} + \alpha(c_{max} - c_{min}) \leq evaluar(G, s_c) \leq c_{max}$ **entonces**
15: $S_{mejores} = S_{mejores} \cup s_c$
16: **fin si**
17: **fin para**
18: $v_{mejor} = seleccion\_uniforme(S_{mejores})$
19: $s_0 = s_0 \cup v_{mejor}$
20: **fin para**
21: **devolver** s_0

Como se puede ver en la línea 14 es necesario tener un procedimiento para evaluar la solución construida hasta ese momento y saber si puede ir a la lista de los mejores posibles de donde posteriormente se seleccionará de manera aleatoria y con probabilidad uniforme uno de los mejores candidatos a ser agregados a la solución. Note que con $\alpha = 1$ el algoritmo es glotón y con $\alpha = 0$ el algoritmo es aleatorio. El procedimiento de evaluación se describe en el Algoritmo 4.2

Posterior a la construcción de la solución inicial, se realiza un procedimiento de búsqueda local. En este caso hay un sólo vecindario viable. Dados dos vértices $u, v \in V$ con $u \neq v$ y tal que el color de u sea negro y v incoloro o viceversa, se hace un intercambio en cuanto al color asignado. Esto no afectará a la solución con respecto al número de vértices negros pero sí requiere evaluar nuevamente la solución. Si la solución después del intercambio es mejor se

Algoritmo 4.2 Función evaluar con procedimiento de reparación de soluciones

Entrada: $G = (V, E)$, s.
Salida: $f_{aptitud}$

```
1:  s.y = ¬s.x
2:  para cada  v ∈ V hacer
3:    si s.y_v = 1 entonces
4:      t = 0
5:      para k ∈ N(v) hacer
6:        t = t ∨ s.x_k
7:      fin para
8:    fin si
9:    s.y_v = ¬t
10: fin para
11: devolver ∑_{s.v∈V} s.y_v
```

cambia y si no, se ignora. Este vecindario es de tamaño $O(|V|^2)$ y se explora en su totalidad. La evaluación de las soluciones también se realiza utilizando el Algoritmo 4.2.

Dado que el GRASP es un método multi arranque otro de los parámetros es cuántos reinicios se permitirán en el procedimiento ($maxI$). Por lo que para la calibración automática, sólo se requiere ajustar α y $maxI$. Claramente α es un flotante entre 0 y 1 y $maxI$ será un entero entre 1 y 100.

La Tabla 4.1 muestra las mejores configuraciones para el GRASP obtenidas con **irace**.

En la Tabla 4.2 se puede ver la mejor solución encontrada para cada instancia mediante la nueva calibración después de 100 corridas contra la original reportada en [6] en las mismas condiciones. El numero de evaluaciones no es posible compararlo puesto que en [6] ese dato no está disponible. Tampoco se reporta tiempo de ejecución, solo se reporta un limite en el tiempo de ejecución de 2 minutos.

El GRASP de [6] propone un valor de $\alpha = 0.9$ además de que no es multi arranque.

Configuración	$maxI$	α
c_1	8	0.7194
c_2	9	0.8618

Tabla 4.1: Mejores configuraciones para GRASP

De los resultados mostrados en la Tabla 4.2 es posible decir que la calibración de parámetros mejora considerablemente la calidad de los resultados obtenidos por el GRASP propuesto

Alg/Instancia	$C_{15} \boxtimes C_{11}$										$C_{21} \boxtimes C_{20}$	
	b=20	**b=21**	**b=25**	**b=28**	**b=70**	**b=99**	**b=100**	**b=120**	**b=125**	**b=141**	**b=80**	**b=100**
GRASP Berend *et al.*	123	120	116	111	71	44	41	21	16	6	300	276
GRASP + Calibración automática	123	121	116	113	71	44	41	22	18	8	300	276

Alg/Instancia	pl_{200}							pl_{240}						
	b=10	**b=25**	**b=50**	**b=75**	**b=100**	**b=125**	**b=150**	**b=10**	**b=25**	**b=50**	**b=75**	**b=100**	**b=125**	**b=150**
GRASP Berend *et al.*	186	168	140	113	87	66	41	186	168	143	119	89	64	41
GRASP + Calibración automática	187	170	146	119	93	70	45	186	170	143	119	91	69	44

Alg / Instancia	$rnd_{200,0.1}$		$rnd_{500,0.1}$			$rnd_{500,0.2}$				$rnd_{600,0.5}$
	b=10	**b=40**	**b=20**	**b=40**	**b=150**	**b=20**	**b=40**	**b=100**	**b=150**	**b=20**
GRASP Berend *et al.*	121	37	149	68	15	290	179	64	36	54
GRASP + Calibración automática	121	38	149	68	15	290	180	65	38	54

Tabla 4.2: Resultados experimentales

por Berend *et al.*, Sin embargo, la comparación contra dicho método es un tanto injusta porque en la referencia no se menciona nada sobre el número de reinicios por tanto se deja solamente indicado que los parámetros obtenidos mediante irace mejoran la calidad de las soluciones obtenidas y por la misma razón no se compara con la búsqueda Tabú ni con los algoritmos genéticos.

4.2. Algoritmo genético para OBWC

En esta sección se mostrará cómo se adapta el algoritmo genético para resolver OBWC. Similar al caso del GRASP representaremos una solución por medio de un vector de tamaño $|V|$ con entradas 0 o 1. Donde x que representa a los vértices que son coloreados con el color negro.

4.2.1. Generación de soluciones iniciales

En la generación de soluciones iniciales (ini_{op}) se crean $POP \in \mathbb{N}$ individuos (soluciones iniciales) con los siguientes métodos:

- Solución inicial aleatoria (uniforme).
- Todas las posiciones de x en 0 (todos 0).
- Todas las posiciones de x en 1 (todos 1).

4.2.2. Función de aptitud

Como en este caso se está resolviendo el OBWC, entonces tenemos un número fijo (b) de vértices de color negro. Se busca encontrar el conjunto de tamaño b que maximice el número de blancos posibles. Suponiendo que el número de negros de una solución s generada por

el AG es b_s. El AG puede en momentos tener soluciones con $b_s > b$ o $b_s < b$ por lo que se debe agregar en la función de aptitud el manejo de esa infactibilidad. También se debe agregar algún mecanismo para el caso de las infactibilidades generadas por la condición de anticoloración. Existen 3 posibilidades.

- Pena de muerte.
- Función de penalización mas un procedimiento de reparación.
- Función de penalización, conteo de todas las infactibilidades.

En la pena de muerte se castigan con un valor muy malo en la evaluación a las soluciones infactibles. Dicho valor es el mismo para todas las soluciones infactibles lo cual limita la búsqueda ya que en caso de estar cerca de la región factible esta decisión aleja al algoritmo de dicha región.

En el caso de una función de penalización mas un procedimiento de reparación. Se emplea el algoritmo 4.2 para colocar color blanco únicamente a los vértices no adyacentes a los vértices negros y posteriormente se utiliza la función de aptitud de la ecuación 4.1.

$$f_{aptitud} = \sum_{v \in V} y_v - |V| \left(\sum_{v \in V} x_v - b \right)^2 \tag{4.1}$$

La razón de la potencia al cuadrado es para que esa parte de la ecuación siempre sea positiva de lo contrario al dar negativo mejoraría la función de aptitud aunque se tratara de una solución infactible en el número de vértices negros. De esa misma parte de la ecuación multiplicarla por $|V|$ garantiza que las soluciones infactibles tengan valor negativo.

El último método viable es quitar el procedimiento de reparación y llevar un conteo de todas las infactibilidades generadas por una solución dada. La ecuación 4.2 refleja dicho conteo de infactibilidades.

$$f_{aptitud} = \sum_{v \in V} y_v - |V| \left[\left(\sum_{v \in V} x_v - b \right)^2 + \sum_{\{u,v\} \in E} (x_u y_v + x_v y_u) + \sum_{v \in V} x_v y_v \right] \tag{4.2}$$

Tras algunas pruebas experimentales, tanto el procedimiento de pena de muerte como el conteo de todas las infactibilidades fueron de pobre desempeño. Por el contrario la función de aptitud de la ecuación 4.1 mas el procedimiento de reparación fueron las que operaron mejor con el algoritmo genético.

4.2.3. Operadores: selección, cruza, mutación y remplazo

Para el operador de selección sel_{op} se tomaron las siguientes posible opciones.

- Se selecciona al mejor individuo de cada generación (rank).

- Se selecciona al individuo mediante ruleta (ruleta).
- Se selecciona al individuo por medio de torneo (torneo).
- Se selecciona al individuo por medio de un muestreo determinista (dss).
- Se selecciona al individuo por medio de un muestreo estocástico (srs).
- Se selecciona al individuo con probabilidad uniforme (uniforme).

El AG desarrollado tomó en cuenta todos los operadores de selección antes mencionados para poder elegir la mejor opción para el problema tratado.

En cuanto al operador de cruza ($cruza_{op}$) se tomaron en cuenta las siguientes opciones para la creación del AG.

- Cruza de un punto (un punto).
- Cruza de dos puntos (dos puntos).
- Cruza uniforme (uniforme).
- Cruza par impar (par impar).

Para el operador de mutación se decidió hacer un simple cambio de bit en la solución. El reemplazo se hace con el mismo operador que se haya elegido para la selección de los individuos a cruzarse, pero en este caso el operador elige entre padres e hijos los que pasarán a la siguiente generación.

4.2.4. Parámetros

Además de decidir qué operadores se pueden utilizar en el AG. Es necesario determinar algunos parámetros que son:

- Probabilidad de cruza ($pc \in [0, 1]$).
- Probabilidad de mutación ($pm \in [0, 1]$).
- Tamaño de la población ($POP \in \mathbb{N}$).
- Número de generaciones ($N_{gen} \in \mathbb{N}$).
- Porcentaje de remplazo ($pr \in [0, 1]$).

Determinar el valor que tendrá cada parámetro, así como qué operadores se utilizarán, se hizo con el procedimiento conocido como F-Race con la herramienta irace [23]. Se realizaron 8000 ejecuciones con los siguientes rangos de variación y tipo de datos.

- Probabilidad de cruza, flotante ($pc \in [0, 1]$).
- Probabilidad de mutación, flotante ($pm \in [0, 1]$).
- Tamaño de la población, entero ($POP \in [10, 150]$).
- Número de generaciones, entero ($N_{gen} \in [100, 50000]$).
- Porcentaje de remplazo, flotante ($pr \in [0, 1]$).

Adicional a los parámetros se requiere un conjunto de instancias de entrenamiento. En este caso se optó por elegir algunas de las instancias más complicadas propuestas en [6] que son las siguientes:

- $C_{11} \boxtimes C_{15}$, $b = 125$.
- $C_{21} \boxtimes C_{20}$, $b = 100$.
- Una gráfica plana con 200 vértices (Pl_{200}) y $b = 150$.
- Una gráfica plana con 240 vértices (Pl_{240}) y $b = 150$.
- Una gráfica aleatoria binomial de 500 vértices con probabilidad de existencia de arista de $p = 0.1$ ($Rnd_{500,0.1}$) y $b = 150$.
- Una gráfica aleatoria binomial de 500 vértices con probabilidad de existencia de arista de $p = 0.2$ ($Rnd_{500,0.2}$) y $b = 150$.

De la calibración mediante irace, se obtuvieron 5 posibles configuraciones de parámetros que se pueden observar en la Tabla 4.3.

Configuración	pc	pm	POP	N_{gen}	sel_{op}	ini_{op}	$cruza_{op}$	pr
c_1	0.4507	0.0079	90	9419	dss	uniforme	par impar	0.4937
c_2	0.4626	0.0078	89	9502	dss	uniforme	par impar	0.4739
c_3	0.5014	0.0033	85	9181	dss	uniforme	par impar	0.4981
c_4	0.5055	0.0033	84	9241	dss	uniforme	par impar	0.4955
c_5	0.4119	0.0039	77	9249	dss	uniforme	par impar	0.4728

Tabla 4.3: Mejores configuraciones de parámetros para el AG después de la ejecución de irace.

4.2.5. Algoritmo genético con tasa de mutación variable

Como se puede ver en la Tabla 4.3, la tasa de mutación es bastante baja por lo que bajo esta evidencia se decidió implementar adicionalmente un algoritmo con tasa de mutación variable. Para ello se utiliza la propuesta de Bäck y Schütz [1] en donde se da una tasa de mutación decreciente en función de la generación en ejecución t, como se puede observar en la ecuación 4.3.

$$pm(t) = \left(2 + \frac{|V| - 2}{N_{gen} - 1} t\right)^{-1} \tag{4.3}$$

Después del ajuste de parámetros vía irace sin tomar en cuenta la mutación ya que varía en cada generación. Las configuraciones de parámetros obtenidos para este AG con tasa de mutación variable ($AG - NM$) se pueden ver en la Tabla 4.4.

Configuración	pc	POP	N_{gen}	sel_{op}	ini_{op}	$cruza_{op}$	pr
c_1	0.6417	132	15674	uniforme	uniforme	dos puntos	0.9325
c_2	0.9406	119	17611	ruleta	uniforme	dos puntos	0.9770
c_3	0.6251	142	15810	uniforme	uniforme	dos puntos	0.9113
c_4	0.8279	117	18970	ruleta	uniforme	dos puntos	0.9836

Tabla 4.4: Mejores configuraciones de parámetros para el $AG - NM$ después de la ejecución de irace.

4.2.6. Resultados Experimentales

Dado que en [6] sólo se muestran las instancias con el mejor resultado obtenido por las meta-heurísticas de dicho artículo siendo la Búsqueda Tabú la que da el mejor desempeño. En nuestro caso, el punto de comparación será que el AG en sus dos versiones alcance la mejor solución encontrada. En algunos casos esta solución coincide con el óptimo. También se resolvió cada una de las instancias con Gurobi, dándole un tiempo de ejecución de 1 hora en una máquina con procesador AMD Opteron(tm) con 24 núcleos y 128 Gb de RAM para fines comparativos. El desempeño de cada algoritmo en cada instancia se hace con la siguiente relación:

$$Q_A = \frac{MEJOR_A}{MEJOR_{CONOCIDO}}.$$

Donde $MEJOR_A$ es la mejor solución encontrada por el algoritmo A y $MEJOR_{CONOCIDO}$ es la mejor solución encontrada hasta el momento. Con ello el nivel de desempeño podrá medirse con valores entre 0 y 1 independientemente de la instancia. La Tabla 4.5 nos muestra en promedio cuantas de las 100 ejecuciones llegaron al mejor valor encontrado para cada tipo de instancia y la Tabla 4.6 nos muestra qué tan buenas fueron las soluciones para cada tipo de instancia, es decir se calcula el Q_A promedio por tipo de instancia.

Alg/Instancia	$C_m \boxtimes C_n$	pl_n	$rnd_{n,p}$
AG	90	90	50
AG-NM	98	95	80
TABU-R	94	57	25
TABU-SG	75	57	75

Tabla 4.5: Comparación de desempeño (número de veces que se alcanzó la mejor solución encontrada)

Alg/Instancia	$C_m \boxtimes C_n$	pl_n	$rnd_{n,p}$
AG	0.99	0.99	0.94
AG-NM	1.0	0.99	0.97
TABU-R	1.0	0.98	0.86
TABU-SG	0.96	0.97	0.64

Tabla 4.6: Comparación de desempeño Q_A promedio por tipo de instancia

Los resultados obtenidos con el AG y el $AG-NM$ después de 100 ejecuciones se muestran en la Tabla 4.7 así como su comparación con la Búsqueda Tabú con solución inicial aleatoria ($TABU-R$) y con la Búsqueda Tabú con solución inicial semiglotona similar a la del GRASP ($TABU-SG$). En la Figura 4.3 se muestra el valor de Q_A para cada heurística comparada. Los promedios y desviaciones estándar de las 100 ejecuciones para cada instancia se muestran en la Tabla 4.8. Cabe destacar que para las dos propuestas de AG el promedio se mantiene alto y la desviación estándar baja, lo cual indica que el AG encuentra buenas soluciones y con poca dispersión lo cual es bueno para una técnica meta-heurística. Las Figuras 4.1 y 4.2 muestran el comportamiento de las ejecuciones de algunas instancias seleccionadas. Es posible notar que el comportamiento del $AG-NM$ es más robusto que el del AG aunque para ello es necesario corroborarlo con alguna prueba estadística como la Wilcoxon.

En este caso se utilizará la prueba Wilcoxon que sirve para comprobar la similitud estadística entre dos conjuntos de datos pareados. La Tabla 4.9 muestra los resultados de dichas pruebas y en base a los datos obtenidos se puede concluir que no hay diferencias estadísticamente significativas entre AG y $AG-NM$ para las instancias seleccionadas.

Alg/Instancia	$C_{15} \boxtimes C_{11}$										$C_{21} \boxtimes C_{20}$	
	b=20	**b=21**	**b=25**	**b=28**	**b=70**	**b=99**	**b=100**	**b=120**	**b=125**	**b=141**	**b=80**	**b=100**
Exacta	123	121	116	113	71	44	42	22	18	8	300	276
Gurobi	123	121	116	113	71	44	41	22	18	8	300	276
$MEJOR_{AG}$	123	121	116	113	71	44	41	22	18	8	298	276
$MEJOR_{AG-NM}$	**123**	**121**	**116**	**113**	**71**	**44**	**41**	**22**	**18**	**8**	**300**	**276**
$MEJOR_{TABU-R}$	123	121	116	113	71	44	41	21	18	8	300	276
$MEJOR_{TABU-SG}$	123	120	116	113	71	44	41	22	16	6	300	276

Alg/Instancia	pl_{200}							pl_{240}						
	b=10	**b=25**	**b=50**	**b=75**	**b=100**	**b=125**	**b=150**	**b=10**	**b=25**	**b=50**	**b=75**	**b=100**	**b=125**	**b=150**
Exacta	≥ 39	≥ 24	NA	NA	NA	NA	NA	≥ 63	≥ 48	≥ 25	NA	NA	NA	NA
Gurobi	187	170	146	119	93	70	**45**	186	170	143	119	91	69	**45**
$MEJOR_{AG}$	187	170	146	119	93	70	44	186	170	143	117	90	69	44
$MEJOR_{AG-NM}$	**187**	**170**	**146**	**119**	**93**	**70**	44	**186**	**170**	**143**	**117**	90	**69**	44
$MEJOR_{TABU-R}$	186	168	144	117	90	68	43	185	170	143	117	**91**	69	**45**
$MEJOR_{TABU-SG}$	**187**	169	143	115	93	65	**45**	186	169	143	**119**	89	69	41

Alg / Instancia	$rnd_{200,0.1}$		$rnd_{500,0.1}$			$rnd_{500,0.2}$				$rnd_{600,0.5}$
	b=10	**b=40**	**b=20**	**b=40**	**b=150**	**b=20**	**b=40**	**b=100**	**b=150**	**b=20**
Exacta	≤ 123	≤ 44	≤ 163	≤ 80	≤ 24	≤ 307	≤ 201	≤ 88	≤ 59	≤ 60
Gurobi	121	39	124	62	24	271	157	68	**45**	43
$MEJOR_{AG}$	121	39	147	62	16	289	176	62	40	50
$MEJOR_{AG-NM}$	**121**	39	**152**	68	**20**	**290**	177	**68**	40	50
$MEJOR_{TABU-R}$	121	39	150	71	17	290	180	58	43	53
$MEJOR_{TABU-SG}$	121	**40**	151	**72**	15	290	**181**	67	39	**55**

Tabla 4.7: Resultados experimentales.

Alg/Instancia	$C_{15} \boxtimes C_{11}$										$C_{21} \boxtimes C_{20}$	
	b=20	**b=21**	**b=25**	**b=28**	**b=70**	**b=99**	**b=100**	**b=120**	**b=125**	**b=141**	**b=80**	**b=100**
Promedio AG	122.78	120.97	114.9	112.16	68.98	39.43	37.97	20.55	17.4	6.65	288.10	262.28
Desv est AG	0.6902	0.1714	1.6667	1.4264	4.6666	4.4044	4.0787	0.9031	2.5346	2.4919	5.5057	7.5746
Promedio AG – NM	122.78	120.18	115.90	112.29	68.74	36.24	35.72	20.89	18.00	5.92	293.96	266.48
Desv est AG – NM	0.6289	0.3775	0.4381	0.9648	4.7313	1.1256	0.9107	0.3145	0.0000	2.5045	5.1930	5.7830

Alg/Instancia	pl_{200}							pl_{240}						
	b=10	**b=25**	**b=50**	**b=75**	**b=100**	**b=125**	**b=150**	**b=10**	**b=25**	**b=50**	**b=75**	**b=100**	**b=125**	**b=150**
Promedio AG	185.46	165.85	137.42	110.71	84.45	61.17	37.27	183.68	164.56	135.25	106.63	81.40	56.39	34.37
Desv est AG	0.7166	1.8333	2.9239	3.4824	2.9555	3.3995	3.0743	1.6568	3.0594	4.4798	4.2727	4.8200	5.0630	3.6698
Promedio AG – NM	186.06	167.27	139.78	113.38	88.64	62.99	38.26	184.71	166.13	137.89	110.82	86.47	61.39	44.03
Desv est AG – NM	0.5472	1.3843	2.7619	2.5217	2.4309	2.6799	1.9417	1.0080	2.2049	3.2377	3.2949	3.4362	4.1936	5.2541

Alg / Instancia	$rnd_{200,0.1}$		$rnd_{500,0.1}$			$rnd_{500,0.2}$				$rnd_{600,0.5}$
	b=10	**b=40**	**b=20**	**b=40**	**b=150**	**b=20**	**b=40**	**b=100**	**b=150**	**b=20**
Promedio AG	119.57	34.57	135.3200	54.4800	12.3200	279.03	160.18	54.33	29.98	40.82
Desv est AG	1.4582	1.9083	4.0995	2.7797	5.9796	5.4206	4.8500	2.9612	2.1648	2.4674
Promedio AG – NM	120.92	36.66	141.96	59.43	16.84	284.43	167.66	57.91	37.54	44.26
Desv est AG – NM	0.3387	1.4580	4.0274	2.9275	1.6497	2.9102	4.2645	2.3872	1.8335	4.4006

Tabla 4.8: Promedios y desviaciones estándar de las dos propuestas de AG.

Instancia	$p - valor$
$C_{15} \boxtimes C_{11}, b = 120$	1.286e-06
$C_{21} \boxtimes C_{20}, b = 100$	6.486e-05
$pl200, b = 100$	0.0
$rnd_{500,0.2}, b = 20$	0.0

Tabla 4.9: Resultados de las pruebas de Wilcoxon.

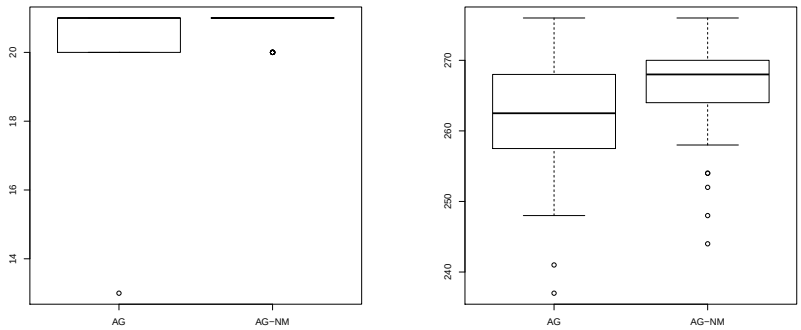

Figura 4.1: Gráficas de caja de las ejecuciones de las instancias $C_{15} \boxtimes C_{11},\ b = 120$ (izquierda) y $C_{21} \boxtimes C_{20},\ b = 100$ (derecha).

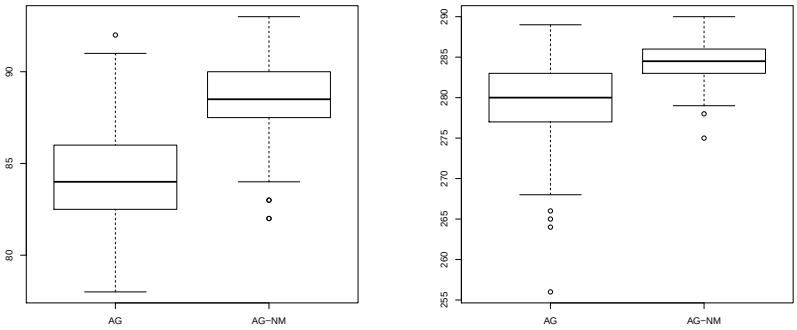

Figura 4.2: Gráficas de caja de las ejecuciones de las instancias $pl200,\ b = 100$ (izquierda) y $rnd_{500,0.2},\ b = 20$ (derecha).

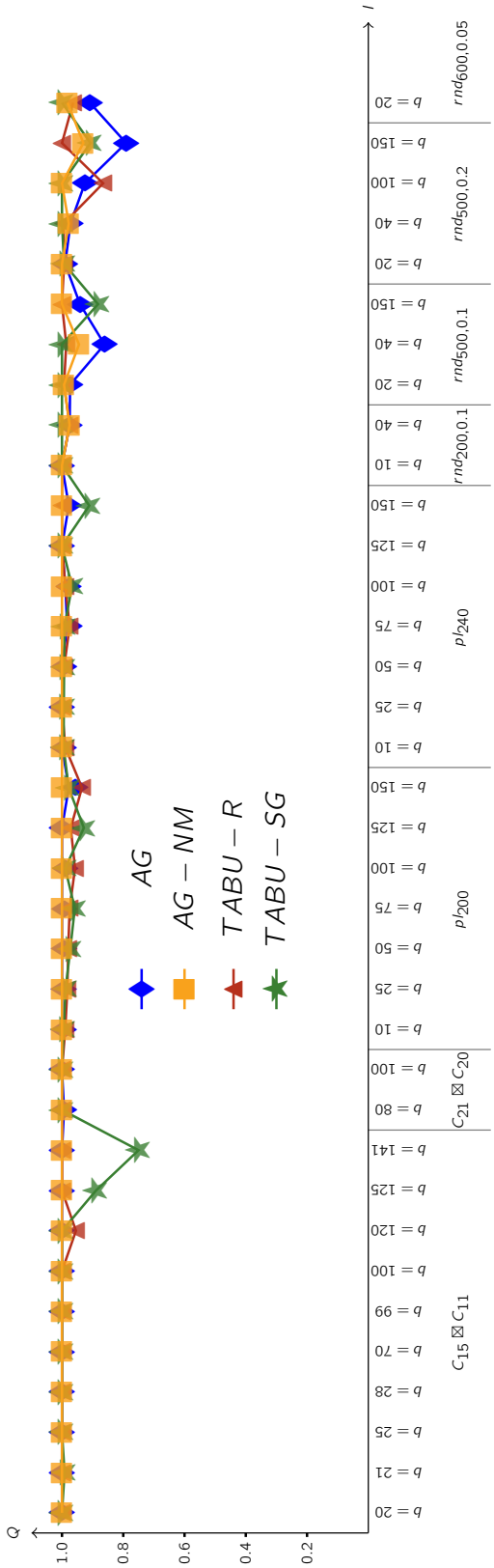

Figura 4.3: Gráfica de resultados experimentales.

A continuación se mencionarán notas sobre el artículo [6]. En el caso de la gráfica $C_{15} \boxtimes C_{11}$ con $b = 141$ se afirma que $w_{opt} = 9$. Como esta gráfica corresponde con los tableros de reyes sobre un toro, entonces podemos usar la fórmula correspondiente del Teorema 2.3.

$$w_{opt} = mn - b - (4a - 4) = 165 - 141 - (4\lceil\sqrt{165 - 141}\rceil - 4) = 8$$

Dado que $mn - a^2 \leq b \leq mn - a(a-1) - 1$ con $a = \lceil\sqrt{mn - b}\rceil$. Por lo tanto se corrige en esa instancia el mejor conocido. Por otro lado sólo queda mencionar que la gráfica $rnd_{500,0.1}$ resulta algo extraña ya que posee menos aristas que $rnd_{500,0.2}$ pero la mejor solución encontrada del caso con menos aristas tiene menos vértices blancos para $b = 150$. Por último en el caso de las gráficas planas se menciona el uso de lp-solve para encontrar la solución "óptima" a través de programación entera, no se menciona si la solución encontrada es la óptima sólo se menciona que es el resultado despues de 1 hora de ejecutar lp-solve. En el caso de este trabajo, se consiguieron las instancias originales provistas por los mismos autores y se alcanzó el óptimo en cuestión de segundos con Gurobi, en algunos casos no coincide el óptimo que se obtuvo en este trabajo con el que ellos reportan. La instancia fue revisada con mucho cuidado y se solicitó de nuevo su reenvió y los resultados fueron los mismos.

4.3. Función de aptitud para tres colores

Para la implementación de meta-heurísticas basadas en el modelo de programación lineal entera (3.67–3.73) es necesario hacer cambios al Algoritmo 4.2 para que permita evaluar soluciones para 3 colores con dos de ellos. El procedimiento se puede ver en el Algoritmo 4.3. Dicho algoritmo funciona asumiendo que los dos primeros colores están bien colocados, es decir la colocación es factible.

En el Algoritmo 4.3 $t_{rojo,azul}$ es un indicador de que existe al menos una arista con terminales rojo y azul, $t_{verde,azul}$ es un indicador de que existe al menos una arista con terminales verde y azul en el vecindario del vértice v y $\downarrow$ es la disyunción lógica negada.

En el caso de que deseemos implementar la función de evaluación con corrección como en el algoritmo genético para OBWC. Existe la posibilidad de que en una solución s el número de vértices rojos $r_s \neq r$. De manera similar con el número de vértices con color verde $g_s \neq g$. Adicional a esto puede suceder cualquiera de las siguientes situaciones. Para cada arista $\{u, v\} \in E$

1. u es rojo y v es verde o viceversa.
2. u es rojo y v es azul o viceversa.
3. u es verde y v es azul o viceversa.

Como podemos ver en el Algoritmo 4.3 asigna el color azul hasta el final por lo que primero se debe comprobar que tanto el rojo como el verde están colocados en lugares factibles. Ello se

Algoritmo 4.3 Función evaluar con procedimiento de reparación de soluciones para 3 colores

Entrada: $G = (V, E)$, s.
Salida: $f_{aptitud}$

```
1:  s.z = s.x ↓ s.x
2:  para cada v ∈ V hacer
3:    si s.z_v = 1 entonces
4:      t_{rojo,azul} = 0
5:      t_{verde,azul} = 0
6:      para k ∈ N(v) hacer
7:        t_{rojo,azul} = t_{rojo,azul} ∨ s.x_k
8:        t_{verde,azul} = t_{verde,azul} ∨ s.y_k
9:      fin para
10:   fin si
11:   s.z_v = t_{rojo,azul} ↓ t_{verde,azul}
12: fin para
13: devolver ∑_{s.v∈V} s.z_v
```

hará con el Algoritmo 4.4. Cabe aclarar que dicho algoritmo no devuelve una solución factible en cuanto al número de rojos (r) y verdes (g).

En el Algoritmo 4.4 hay un componente aleatorio. Es posible quitar dicho componente de dos formas. La primera es decidiendo que siempre se le va a quitar un color en particular en las infactibilidades, ya sea siempre rojo o siempre verde. Por otro lado esto puede provocar que la restricción con respecto al número preestablecido de rojos o verdes se vea alterada. Por lo tanto la segunda propuesta va en el sentido en que se mantenga el color en donde dicha restricción este más cerca de la factibilidad. Eso se puede medir con la diferencia entre el número de rojos o verdes del problema (r o g) y el número de rojos o verdes de la solución actual (r_s o g_s). El procedimiento se puede ver reflejado en el Algoritmo 4.5.

Posterior a la aplicación del verificador de factibilidad y la función de reparación para asignar vértices azules. Se procede a evaluar la solución mediante la siguiente función de aptitud 4.4.

$$f_{aptitud} = \sum_{v \in V} z_v - |V| \left[\left(\sum_{v \in V} x_v - r \right)^2 + \left(\sum_{v \in V} y_v - g \right)^2 \right] \tag{4.4}$$

4.4. Propuestas multi-objetivo

Con respecto al caso multiobjetivo es posible modelar el PAC como un problema de optimización multiobjetivo basándonos en el programa no lineal entero (3.95–3.97). El problema multiobjetivo consta de $k + 1$ funciones objetivo en donde las primeras k funciones objetivo

Algoritmo 4.4 Verificador de factibilidad

Entrada: $G = (V, E)$, s.
Salida: $s_{factible}$

```
1: para cada v ∈ V hacer
2:   si s.y_v = 1 ∧ s.x_v = 1 entonces
3:     p = Aleatorio_Entero_Entre(0, 1)
4:     si p = 0 entonces
5:       s.y_v = 0
6:     si no
7:       s.x_v = 0
8:     fin si
9:   fin si
10: fin para
11: para cada {u, v} ∈ E hacer
12:   si s.x_u = 1 ∧ s.y_v = 1 entonces
13:     p = Aleatorio_Entero_Entre(0, 1)
14:     si p = 0 entonces
15:       s.x_u = 1
16:       s.y_v = 0
17:     si no
18:       s.x_u = 0
19:       s.y_v = 1
20:     fin si
21:   fin si
22: fin para
23: s_factible = s
24: devolver s_factible
```

corresponden con el número de vértices que deben tener asignado el color i con $i \in \{1 \dots k\}$. El modelo multiobjetivo queda como sigue:

$$z = \left(\max \sum_{v \in V} x_{v,1}, \dots, \max \sum_{v \in V} x_{v,k}, \min F \right) \tag{4.5}$$

$$x_{v,i} \in \{0, 1\} \text{ para toda } v \in V \text{ e } i \in \{1, \dots, k\}. \tag{4.6}$$

De donde

$$F = \sum_{i,j \in \{1 \dots k | i \neq j\}} \sum_{\{u,v\} \in E} (x_{u,i} x_{v,j} + x_{v,i} x_{u,j}) + \sum_{i,j \in \{1 \dots k | i \neq j\}} \sum_{v \in V} x_{v,i} x_{v,j}$$

Una manera de resolver el problema multiobjetivo es mediante el método de restricciones ε. El procedimiento se puede ver en el Algoritmo 3.1. De esta manera F dejaría de ser un

Algoritmo 4.5 Verificador de factibilidad versión 2

Entrada: $G = (V, E)$, s, r, g.
Salida: $s_{factible}$

1: $r_s = \sum_{v \in V} s.x_v$
2: $g_s = \sum_{v \in V} s.y_v$
3: **para cada** $v \in V$ **hacer**
4: **si** $s.y_v = 1 \wedge s.x_v = 1$ **entonces**
5: **si** $r_s - r < g_s - g$ **entonces**
6: $s.y_v = 0$
7: $g_s = g_s - 1$
8: **si no**
9: $s.x_v = 0$
10: $r_s = r_s - 1$
11: **fin si**
12: **fin si**
13: **fin para**
14: **para cada** $\{u, v\} \in E$ **hacer**
15: **si** $s.x_u = 1 \wedge s.y_v = 1$ **entonces**
16: **si** $r_s - r < g_s - g$ **entonces**
17: $s.x_u = 1$
18: $s.y_v = 0$
19: $g_s = g_s - 1$
20: **si no**
21: $s.x_u = 0$
22: $s.y_v = 1$
23: $r_s = r_s - 1$
24: **fin si**
25: **fin si**
26: **fin para**
27: $s_{factible} = s$
28: **devolver** $s_{factible}$

objetivo y pasaría a ser una restricción no lineal entera como se muestra en el programa (4.7–4.9).

$$z = \left(\text{máx} \sum_{v \in V} x_{v,1}, \ldots, \text{máx} \sum_{v \in V} x_{v,k} \right) \quad (4.7)$$

$$\text{mín} \sum_{i,j \in \{1 \ldots k | i \neq j\}} \sum_{\{u,v\} \in E} (x_{u,i} x_{v,j} + x_{v,i} x_{u,j}) + \sum_{i,j \in \{1 \ldots k | i \neq j\}} \sum_{v \in V} x_{v,i} x_{v,j} \leq \varepsilon \quad (4.8)$$

$$x_{v,i} \in \{0,1\} \text{ para toda } v \in V \text{ e } i \in \{1 \ldots k\}. \quad (4.9)$$

Claramente para que el programa (4.7–4.9) corresponda con un problema de anticoloración, $\varepsilon = 0$.

Si se intenta resolver el problema de manera directa con únicamente dos colores, ya sea por una escalarización simple, es decir una suma ponderada de los objetivos o resolviendo de manera lexicográfica dando prioridad a alguno de los objetivos. Las únicas soluciones que se encuentran de esa manera son $(|V|, 0)$ y $(0, |V|)$. Ambas soluciones son no dominadas pero representan a las soluciones triviales únicamente.

Existen en la literatura distintas maneras de escalarización diferentes a la suma ponderada que pueden darnos una mejor aproximación al frente de Pareto. Para un estudio a profundidad de los métodos de escalarización se recomienda consultar [13].

En nuestro caso nos basaremos en el uso de la norma p que se muestra en la ecuación 4.10.

$$\min_x \left(\sum_{i=1}^{k} \omega_i \| z_i^* - f_i(x) \|^p \right) \quad (4.10)$$

Donde $\| z_i^* - f_i(x) \|^p = \text{máx}_{i=1}^{k} (z_i^* - f_i(x))^{\frac{1}{p}}$ es la norma p y ω_i es un vector de pesos, tal que $\sum_{i=1}^{k} \omega_i = 1$. Cuando $p = \infty$ se conoce como la escalarización de Chebyshev y se escribe como sigue.

$$\min_x \{ \max_{i=1 \ldots k} \omega_i (z_i^* - f_i(x)) \} \quad (4.11)$$

Donde z_i^* es conocido como vector ideal que se logra cuando se maximiza cada uno de los objetivos y se obtiene un vector con el "óptimo" de cada objetivo. En nuestro caso $z^* = (|V|, |V|, \ldots, |V|)$. La ecuación 4.11 minimiza la distancia más grande al vector ideal. En la Figura 4.4 se puede ver una representación gráfica para la escalarización de Chebyshev con dos colores.

El modelo de dos colores con escalarización de Chebyshev queda como sigue.

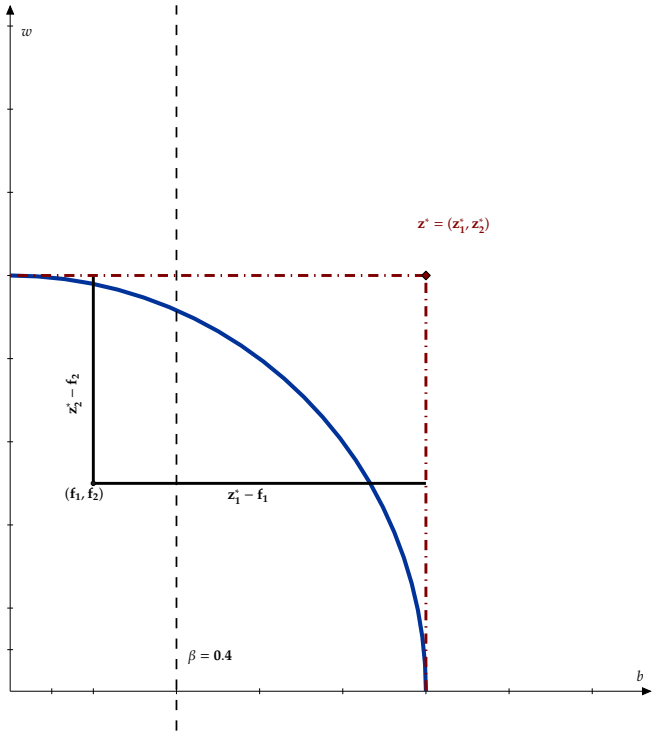

Figura 4.4: Escalarización de Chebyshev

$$\begin{aligned}
\text{mín} \quad & z && (4.12)\\
z \quad & \geq \quad \beta(|V| - b) && (4.13)\\
z \quad & \geq \quad (1-\beta)(|V| - w) && (4.14)\\
\sum_{v \in V} x_v \quad & = \quad b && (4.15)\\
\sum_{v \in V} y_v \quad & = \quad w && (4.16)\\
\sum_{\{u,v\} \in E} (x_u y_v + x_v y_u) \quad & + \quad \sum_{v \in V} x_v y_v \leq 0 && (4.17)\\
x_v, y_v \in \{0,1\}, \ b, w \quad & \in \quad \mathbb{Z}^+ && (4.18)
\end{aligned}$$

Con $\beta \in [0,1]$. Para poder comparar el método con un frente de Pareto que se pueda calcular de manera exacta, se calcula el frente real de la gráfica $C_{11} \boxtimes C_{15}$ que corresponde con los tableros toroidales de reyes. La comparación del frente real y el calculado con la escalarización de Chebyshev se puede observar en la Figura 4.5. Por otro lado en la Figura 4.6 se puede observar un frente aproximado de una gráfica aleatoria con 200 vértices y una probabilidad de 0.1 de existencia de aristas.

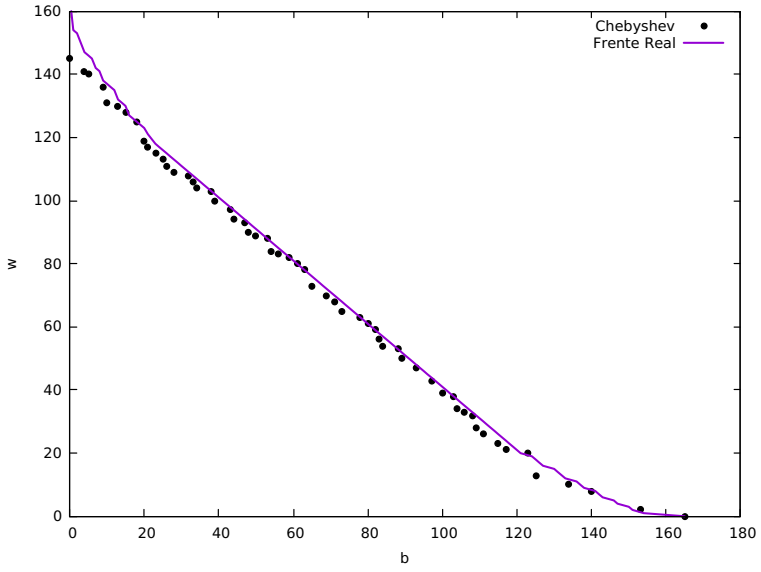

Figura 4.5: Comparacion del frente real y mediante el modelo (4.12–4.18) de la gráfica $C_{11} \boxtimes C_{15}$.

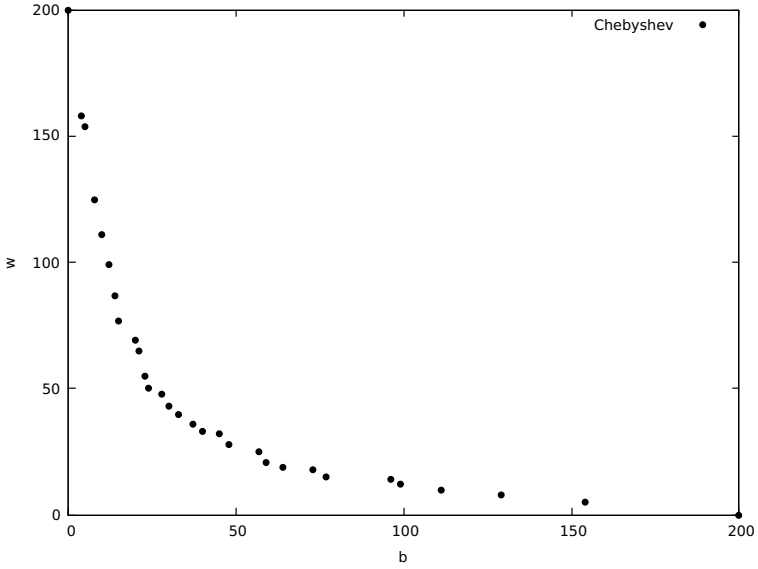

Figura 4.6: Frente aproximado mediante el modelo (4.12–4.18) de la gráfica $rnd_{200,0.1}$.

Conclusiones y trabajo futuro

En este trabajo se contribuyó con distintos modelos para la resolución del PAC con dos o más colores. Adicionalmente se aporta una familia infinita de un caso que se puede anticolorear con dos colores de manera eficiente, en concreto se estableció un teorema para tableros de caballos que permite dar el valor óptimo y la forma de la solución sin necesidad de resolver el problema con alguna metodología exacta, el problema de las reinas continúa como un problema abierto en cuanto a su complejidad. Para más de dos colores, se dio el método por etapas que hasta ahora no se puede decir que es un algoritmo, puesto que no se puede garantizar la optimalidad de las soluciones para gráficas generales, aunque un posible camino a seguir es encontrar familias de gráficas que sí permitan esa propiedad y las candidatas inmediatas son las gráficas planas. En cuanto a los métodos heurísticos es un campo muy poco explorado, por lo que el algoritmo genético es una contribución significativa, sobre todo en el tiempo de ejecución ya que se alcanzaron mejores resultados en un tiempo significativamente menor. En este sentido se puede continuar esta línea de investigación probado algunas otras técnicas. En relación a la optimización multiobjetivo se propuso el uso de una escalarización diferente a la suma ponderada. En la literatura existen una gran variedad de escalarizaciones que pudiesen ser comparadas. El trabajo futuro se puede resumir en las siguientes lineas

Algoritmos exactos

- Demostrar la complejidad del BWC en gráficas bipartitas.
- Enfoque poliedral
- Encontrar familias de gráficas donde el método por etapas pueda resolver de manera exacta.
- Encontrar mejores cotas para el valor de κ en el caso de los tableros balanceados de reinas.
- Resolver el caso no balanceado en reinas.
- Resolver el caso multicolor con reyes, caballos, alfiles y reinas.

Métodos heurísticos

1. Proponer (meta)heurísticas para el caso multicolor.
2. Abordar el caso multicolor con distintos métodos de escalarización.
3. Generar un conjunto de instancias para el caso multicolor.

Para finalizar es deseable encontrar un campo de aplicación del problema. Originalmente se pensaba la seguridad química como un primer campo de aplicación, pero existen distintos fenómenos que podrían prestarse a ser modelados como un problema de anticoloración. Por ejemplo para mantener grupos sociales con peligro de revuelta distanciados (sin implicar un apartheid) o bien en el diseño de circuitos electrónicos.

Bibliografía

[1] T. Bäck and M. Schütz. Intelligent mutation rate control in canonical genetic algorithms. *Foundations of Intelligent Systems*, pages 158–167, 1996.

[2] D. Berend, E. Korach, and O. Yahalom. Anticoloring of the rook's graph. *Discrete Applied Mathematics*, 188:1–15, 2015.

[3] D. Berend, E. Korach, and S. Zucker. Two-anticoloring of planar and related graphs. In *International Conference on Analysis of Algorithms DMTCS proc. AD*, volume 335, page 342, 2005.

[4] D. Berend, E. Korach, and S. Zucker. A reduction of the anticoloring problem to connected graphs. *Electronic Notes in Discrete Mathematics*, 28:445–451, 2007.

[5] D. Berend, E. Korach, and S. Zucker. Anticoloring of a family of grid graphs. *Discrete Optimization*, 5(3):647–662, 2008.

[6] D. Berend, E. Korach, and S. Zucker. Tabu search for the bwc problem. *Journal of Global Optimization*, 54(4):649–667, 2012.

[7] D. Berend and S. Zucker. The black-and-white coloring problem on trees. *J. Graph Algorithms Appl.*, 13(2):133–152, 2009.

[8] C. Berge. *Graphs and hypergraphs*, volume 7. North-Holland publishing company Amsterdam, 1973.

[9] M. Berkelaar. lp-solve.

[10] M. Birattari. The race package for r. racing methods for the selection of the best. 2003.

[11] C. A. Coello, G. B. Lamont, D. A. Van Veldhuizen, et al. *Evolutionary algorithms for solving multi-objective problems*, volume 5. Springer, 2007.

[12] F. COIN-OR. Coin-or.

[13] G. Eichfelder. *Adaptive scalarization methods in multiobjective optimization*, volume 436. Springer, 2008.

[14] U. Elsner. Graph partitioning-a survey. 1997.

[15] T. A. Feo and M. G. Resende. Greedy randomized adaptive search procedures. *Journal of global optimization*, 6(2):109–133, 1995.

[16] M. Garey and D. Johnson. Computers and intractability: a guide to the theory of np-completeness. *WH Freeman & Co., San Francisco*, 1979.

[17] F. Glover. Tabu search—part i. *ORSA Journal on computing*, 1(3):190–206, 1989.

[18] F. Glover. Tabu search—part ii. *ORSA Journal on computing*, 2(1):4–32, 1990.

[19] I. Gurobi Optimization. Gurobi optimizer reference manual, 2016.

[20] P. Hansen, A. Hertz, and N. Quinodoz. Splitting trees. *Discrete Mathematics*, 165:403–419, 1997.

[21] J. H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1992.

[22] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

[23] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, T. Stützle, and M. Birattari. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 2016.

[24] K. Murty. Operations research: Deterministic optimization models, 1995.

[25] C. I. of Mathematics. Millennium problems, 2000.

[26] S. Zucker. The black-and-white coloring problem on chordal graphs. *J. Graph Algorithms Appl.*, 16(2):261–281, 2012.